University of Padova

Department of Physics and Astronomy "Galileo Galilei"

Master Thesis in Physics of Data

Activities Prediction of Business Process Instances using Deep Learning Techniques

Supervisor
Prof. Marco Zanetti
University of Padova

Co-supervisor
Pasqualino Di Nobile
Matteo Greselin

Master Candidate
Lucia Depaoli

Student ID
2016960

Academic Year
2022-2023

"Know how to solve every problem that has been solved"
— Richard P. Feynman

Abstract

The ability to predict the next activity or attributes of an ongoing case is becoming increasingly important in today's businesses. Processes need to be monitored in real-life time to predict the remaining time of an open case, or to be able to detect and prevent anomalies before they have a chance to impact the performances. Moreover, financial regulations and laws are changing, requiring companies' processes to be increasingly transparent. Process mining, supported by deep learning techniques, can improve the results of internal audit activities. The task of predicting the next activity can be used to point out traces at risk that need to be monitored. In this way, companies are aware of the current state of operations and can take resolution actions in time. In recent years, this problem has been tackled using deep learning techniques, such as Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) neural networks, achieving consistent results. The first main contribution of this thesis consists of a generation of a process mining dataset based on the Purchase-to-Pay (P2P) process. The SAP tables structure is taken into account since it is the most popular management software in today's companies. By introducing anomalies, the simulated dataset can be seen as a realistic representation of a company's operational activities. The second contribution of the thesis is an investigation of deep learning techniques that exploit information from both temporal data and static features, applied to the previously generated dataset. The neural networks are then used to predict future events characteristics of running traces. Finally, we discuss real-life application of the results and present future work proposals.

Contents

Listing of figures

Listing of tables

Listing of acronyms

NLP . . . . . Natural Language Processing

BPMN . . . . . Business Process Model and Notation

UML . . . . . Unified Modeling Language

EPC . . . . . Event-driven process chain

KPI . . . . . Key Performance Indicator

SVM . . . . . Support Vector Machine

PCA . . . . . Principal Component Analysis

t-SNE . . . . . t-Distributed Stochastic Neighbor Embedding

AUC-ROC . . . . . Area Under the Curve - Receiver Operator Characteristic

RNN . . . . . Recurrent Neural Network

LSTM . . . . . Long Short-Term Memory

GRU . . . . . Gated Recurrent Unit

seq2seq . . . . . Sequence-to-Sequence

seq2vec . . . . . Sequence-to-Vector

P2P . . . . . Procure-to-Pay or Purchase-to-Pay

PO . . . . . Purchase Order

PR . . . . . Purchase Requisition

ERP . . . . . Enterprise Resource Planning

1 Introduction

In recent years, companies have faced a series of challenges that have highlighted their vulnerability [1]. One major event that had a profound impact on companies was the COVID-19 pandemic. The outbreak of the virus resulted in global restrictions and lockdown measures, forcing many businesses to reduce their production chains and operations [2]. This disruption had far-reaching consequences, affecting industries across various sectors. Additionally, the recent raw material crisis caused meaningful breakdowns for many industries that rely on specific resources for the production [3]. This situation also led to an increase in energy price that resulted in high expenses for manufactures [4], which have proven to be unattainable for many small businesses. These challenges have underscored the need for companies to adapt and find innovative solutions to mitigate the risks associated with such disruptions.

In addition to the challenges faced by companies, in recent years there has been a growing demand for a different kind of information. Customers and society have become more conscious of the environmental impact of the products they consume [5] [6] [7]. What is the journey of the items we buy at the supermarket and how much is its environmental impact? Is this business resilient enough to face unexpected events, such as COVID-19 pandemic? How efficient is the customer service of this company? Adaptation to changing demands, environmental consciousness, business resilience, and customer service are essential considerations for companies that aim to thrive in the current dynamic business landscape. In this framework, emerging technologies such as process mining can provide valuable support on this journey of transformation and optimization [8] [9] [10].

Modern supply chains are characterized by such a complex structure that often precludes the possibility to understand their overall organization. A large company includes numerous processes involving hundreds of different entities and enterprise systems [11]. Generally, businesses handle information exchange through email, paper sheets or using several systems that are often not directly interacting with each other. The risk of information losses as well as imputation errors is high and solving them can be complex and time-consuming [12]. In addition to internal anomalies, extreme events such as earthquakes, global crises, droughts, price increases, can put the entire supply chain at risk [13].

Nowadays, every company records data about their processes. This information is usually stored in large databases and used to make high-level analysis, such as retrieving the total number of orders and their net value over a span time. Despite the amount of data available, companies have not yet evolved into *data-driven* organizations [14]. Using process mining platforms it is possible to get a clearer view of business processes so as to investigate their bottlenecks, and improving and reorganizing current processes [15] [16] [17] [18]. Moreover, coupling process mining with deep learning techniques allows to predict the progress of ongoing processes [19] [20] [21], or to simulate processes following changes [22] [23] [24]. These technologies have been shown to provide numerous benefits, including lowering production costs and time, increasing product reliability, reducing manual workload and decreasing money losses due to data inconsistencies.

This work focuses on the use of deep learning techniques applied to process mining for predictive monitoring problems. Specifically, it is shown how these tools can be used to prevent anomalies in open processes, by anticipating possible failures. First, we provide an introduction to process mining as a powerful tool to improve the performance of companies. Secondly, the focus is shifted towards process prediction, exploring the latest advancements in the field of deep learning techniques. To facilitate the experimentation and evaluation of process prediction models, we introduce a dataset specifically designed. The dataset is a realistic representation of a company's operational activities, enabling experiments to analyze and compare different prediction models. Then, we apply the methodologies introduced to the dataset, comparing the results obtained. We discuss the performance of difference deep learning models the prediction task and highlights any notable findings or insights. Lastly, we conclude with a discussion on future work that outlines potential avenues for further research and development in the field of process prediction and suggests areas where improvements can be made.

1.1 Process Mining

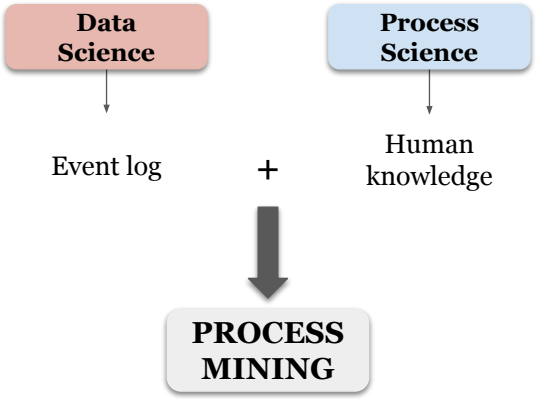

Figure 1.1: Process mining acts as a bridge between data science and process science. It applies the human knowledge to add details to the event log and it derives information about the actual state of processes directly from business data (event log).

Process mining is a process management technique that aims to support business processes through the analysis of the event log. As shown in Fig. 1.1, it represents a bridge between *data science* and *process science*, since the former does not take the process into account, whereas the latter tends to be model-driven [25, Chapter 1]. Process mining seeks to use data to improve end-to-end processes. The core of process mining is the *event log*, which is a structured ordered database that contains information about every activity that happened in the process.

We take as an example a retailer company, which receives product orders and ships the goods to the buyer. The ideal process of each purchased item is the one shown on the left panel of Fig. 1.2, where each item goes through a series of planned activities without encountering any issues. But real-life processes contain anomalies. For example, the buyer may not have enough money in his bank account to proceed with payment, or the requested product may not be in stock. A large number of variables can interfere with the ideal process and can lead to different and sometimes unwanted scenarios, such as the one shown on the right panel of Fig. 1.2. Process mining target is to reconstruct, analyze and understand the real process based on the available data, rather than on the ideal designed process [25, Chapter 2].

Companies often have an overall understanding of their ideal internal processes, but they may lack in awareness of all possible variants that can occur. As a result, they tend to address problems times to times without establishing a systematic approach or without investigating the root causes of the anomalies [26]. However, business data contains valuable information.

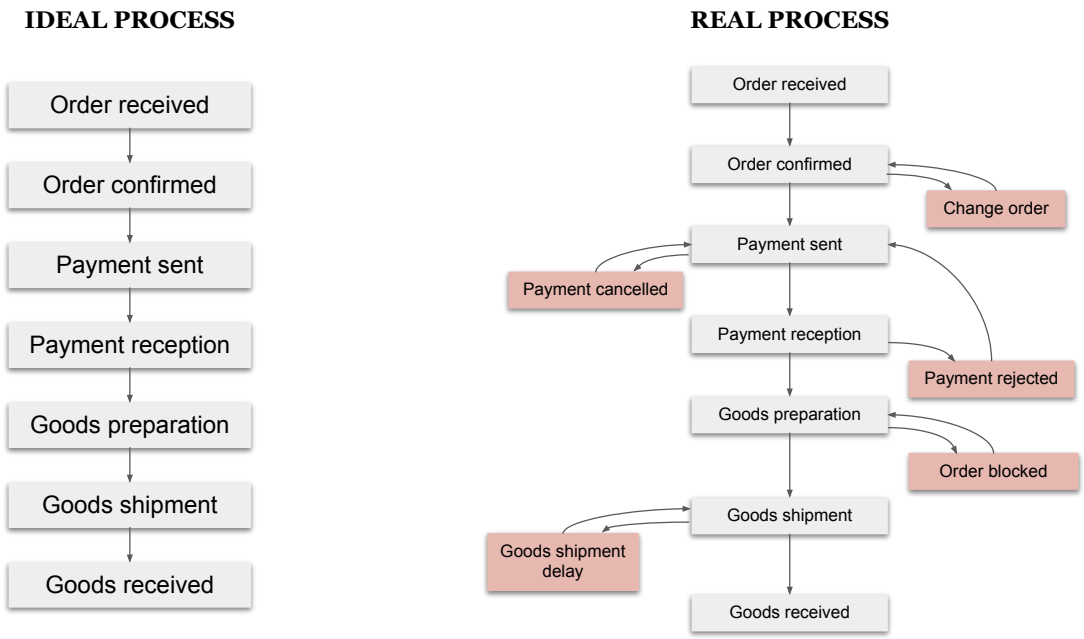

Figure 1.2: Example of a company process. Process on the left shows the ideal course of an order, whereas on the right it is shown the same process but of an order which encounters several anomalies (marked in red) along the way, that lead to different reworks.

Process mining techniques help relate event data to the process model, allowing the companies to discover the actual processes, and to evaluate and enhance the existing ones.

Process mining can exploit its full potential when used in real time, rather than being applied on past and completed events [25, Chapter 10]. This is referred as *operational support*. An example of its application is *auditing*, which refers to the evaluation of organizations and their processes. Audit activities check whether business processes are executed within certain boundaries set by managers, governments, and other stakeholders [27]. By applying process mining techniques in an operational support way, we can detect inconsistencies as soon as they occur, enabling real-time resolution and mitigating risks.

Compared to data mining and machine learning techniques, process mining is based on the fact that its instances are end-to-end processes. Inside each singular cases, events are correlated to each other in a sequential way. It is possible to use data mining and machine learning techniques to support process mining, but they need to be adapted in order to be used with the different data structures. Many similarities with the natural language processing (NLP) field exist, but substantial differences remain [28], such as the vocabulary size, the length of traces, and the presence of overlapping activities.

One of the main challenges of process mining research is the lack of datasets to evaluate performances on [29]: companies do not make their data available because they contain transactional data, customers' personal information and other confidential records. Moreover, companies' data are often of low quality: they contain inconsistencies, missing fields, and formatting errors. As a result, apply deep learning algorithms to process mining data is non trivial. First it is necessary to perform various preprocessing and cleaning operations that require expert support [30] [31].

1.1.1 Life-cycle of a Process Mining project

Typically, every process mining project goes through a series of well-defined steps before it succeeds in making improvements in companies. Projects can run for years and require the support of several figures with different skill sets, such as data analysts, business experts, IT architects. One of the most important figures is the *process owner*, who should be able to describe the high-level process, the supporting information systems, the transactions involved in the databases, and the critical points in the process. Data extraction and transformation becomes difficult if one is not fully familiar with the systems in use. Therefore, *data experts* must support the project since each company uses different set of systems and platforms, with custom field labels and custom data types. *Business experts*, on the other hand, can identify which changes to implement for process improvement and standardization, as well as ensure compliance with the latest regulations.

Usually, a process mining project goes through the following stages [32]:

- *Planning phase*: identification of the objectives of the projects and the questions that need to be answered. During this phase, we take a first look at the data available and understand which of them are useful for the purpose of the investigation and how are they related to each other.
- *Extraction & Transformation phase*: extraction of the chosen data from the source systems and transformation in the target format. This phase can be very time-consuming and wasteful, as there is a large amount of data available to companies and most of it is not used for the project purpose. Moreover, each company has its own data type format and its own support systems. During this phase we define the relationship between the tables and we create the event log.
- *Discovery phase*: using process mining discovery techniques, we create the data model from the event log without any a-priori information. This is one of the key steps of the project, as the data model represents the true/ideal track that events follow.

- *Conformance phase*: comparison of the event log with an existing process / model. This phase aims to check if what happens in reality conforms the ideal process and vice versa.
- *Enhancement phase*: using information hidden in the event log, we investigate possible improvements and eventually we implement them. These can be, for example, a change in the process model to make it more similar to what actually happens, or the implementation of an automatism.

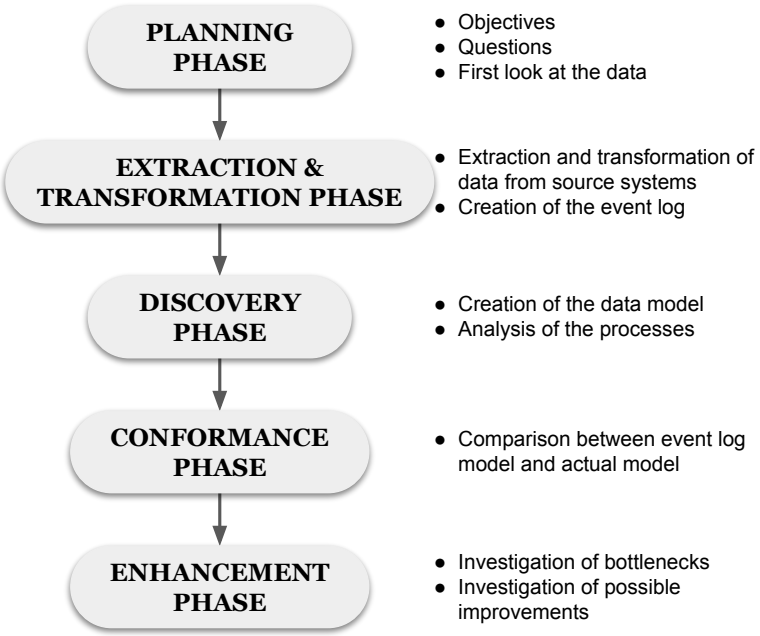

Figure 1.3: Life-cycle of a process mining project

The last three steps represent the core of process mining. During the discovery phase, we can visualize the as-is process and discover bottlenecks that cause slowdowns or productivity losses. For example, we can estimate the percentage of rework, i.e., instances that require duplicate working and thus that cause delays. The conformance phase can be faced using three different approaches, as shown by Wil Van der Aalst [33]. An example of the first approach is the *footprint*, which is basically a matrix containing relationships and dependencies between events. The second approach is based on the concept of *replay*, which means replicating instances of the event log on the model basic, to see which traces are allowed by the model (*normative* model), or how well the model is able to describe what is happening in the systems (*descriptive* model). The third possible approach consists on computing an optimal *alignment* between each trace

and a model allowed path. In the enhancement phase, we take into account the results obtained in the previous stages to make changes. For example, we can modify the process if we observed that it is unable to replicate most of the observed traces, or we can make changes to solve bottlenecks or delays.

1.1.2 Event Log

The main ingredient of process mining is the event log, which is a table containing process instances and their attributes. In order to exist, an event log must consist of at least 2 columns, which are the *cases* column and the *event* column. The first one refers to the ID of process cases, whereas the second contains the name or the code of the recorded activity / event. It is usually present a third column which refers to the events *timestamp*. An example of a simple event log is shown in Tab. 1.1.

CASE ID	ACTIVITY	TIMESTAMP
1	A	12-12-2022
1	B	13-12-2022
1	C	16-12-2022
1	D	17-12-2022
2	A	12-12-2022
2	D	15-12-2022
2	C	20-12-2022

Table 1.1: Example of an event log. In this case we have two process instances (1 and 2), four possible activities (A, B, C, D) and the timestamps of the activities. Case IDs must be unique and must refers to a single instances of the process.

Each row of the event log refers to a single *event*, which is composed of several attributes, such as *case ID*, *activity name* and *timestamp* of the activity. Events are ordered in a sequential way and case IDs must refer to a single instance of the process, since they represent single data points that flow through the process. Process mining techniques analyze the traces present in the event log in order to discover the process directly from the data coming from the systems, rather than analyzing the ideal process, which is often not representative.

In order to be able to reason about event logs and to precisely specify the requirements for them, we formalize various notations [25, Chapter 5]:

Definition 1.1.1 (Event)**.** An event $e \in \mathscr{E}$ is a tuple $e = (c, a, F)$ where $a \in \mathscr{A}$ is the name of the activity, $c \in \mathscr{C}$ is the case ID and F is a set of attributes of the event, such as the timestamp, the person who executed the activity, the cost, the duration, etc. $\mathscr{E}$ represents the *event universe* i.e., the set of all possible events.

In Tab. 1.1, each event is a tuple made by $(case\_id, activity\_name, timestamp)$.

Each event can have various attributes associated to it, for example the starting and ending *timestamp* or the name of the resource that executed it. We define the event attributes as follows:

Definition 1.1.2 (Event attribute)**.** Let $n \in AN$ be an attribute over the set of attribute names. We denote as $\#_n(e)$ the value of attribute n for event e.

For example, a set of possible event attributes is:

- $\#_{timestamp}(e)$: timestamp associated to event e.
- $\#_{resource}(e)$: resource associated to event e.
- $\#_{cost}(e)$: cost associated to event e.

The sequence of the activity for each case is called *trace*, defined as follows:

Definition 1.1.3 (Trace)**.** Let $\mathscr{E}^*$ be the set of all possible sequences over $\mathscr{E}$. A *trace* $t = \langle e_1, ..., e_n \rangle \in \mathscr{E}^*$, is a sequence of events, or a *process instance*.

Traces that follow the same sequence of process activities belong to the same *process variant*. Each variant differs from the others in at least one activity execution. Process variants can also represent a single process instance.

Each trace belongs to a single *case*, which can have attributes:

Definition 1.1.4 (Case attribute)**.** Let $\mathscr{C}$ be the *case universe*, i.e., the set of all possible cases. We denote as $\#_n(c)$ the value of attribute n for case c.

The difference between event attributes and case attributes is that the former vary along the trace as they refer to a single event, while the latter are fixed for each trace. An example of possible case attributes are:

- $\#_{country}(c)$: country associated to case c.
- $\#_{material}(c)$: material associated to case c.
- $\#_{priority}(c)$: priority associated to case c.

Finally, we define the *event log* as the set of traces $T \subseteq \mathscr{E}^*$:

Definition 1.1.5 (Event log)**.** An event log is a set of traces $T \subseteq \mathscr{E}^*$ such that each event appears at most once in the entire event log.

It is important to observe that the event log is not present within the enterprise databases, but it is built starting from them. Experts (process owners, data scientists, etc.) have to identify the main process activities, which may or may not be already listed in an process model. Then, for each activity one must associate the database fields that identify the occurrence of the activity. For example, the activity "Creation Order" is associated with an "Order ID" and the "Creation Date" field, which may be found in different tables. Moreover, each activity can then be associated with attributes, which can be hundreds. In addition to the creation of the activities, it is necessary to construct the Case ID field, which is the process instance identifier. We usually create it by connecting several fields, for example "Country" - "Order ID" - "Invoice Number". This ensures that each case is unique, since different tables extracted from different systems may have duplicate names that do not refer to the same process instance.

1.1.3 Process Model

Since the event log provides the data point of view, the process model highlights a process-centric view. The purpose of process models is to document and provide process knowledge. Even if the ideal process does not represent what really happens, there are some information that are fundamental for the process but may not be written in the data. For example, in a particular process, activity B must be always follow activity A, and activity C is not allowed if activity D has happened. Or also, activity A needs to be always performed by user 1001. The event log does not explicitly contains these information, as this is very likely to contain rule violations, yet these must be taken into account when conducting process analysis. A hand-made process model can be translated into machine language using some particular notations for model operational processes, such as Petri Net, BPMN, UML, and EPCs. The aim of these notations is to allow the visualization of processes and procedures in a standard and user-friendly representation [25, Chapter 3]. Usually process models include the following elements:

- Events: events that starts, changes or completes a process, such as message, timer, error, signal, etc.
- Activities: activities performed by the system or by a person.
- Decision points: points that shape the process according to conditions or events.

- Pools and lanes: represent stakeholders in the process, such as a particular person or a department.

A process model can also include additional information such as the duration of a subprocess or activities attributes. An example of BPMN model is shown in Fig. 1.4.

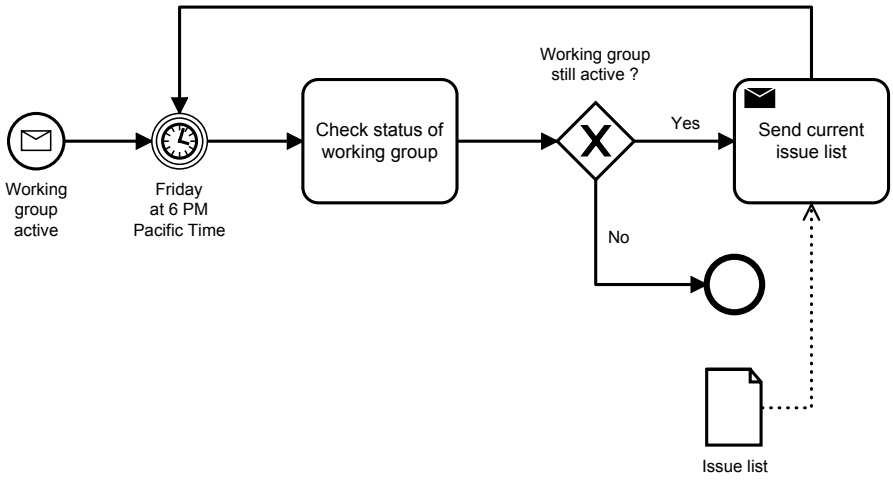

Figure 1.4: Example of a BPMN process model. We can identify the starting event "Working group active", followed by an intermediate timer event. Next, we have the activity "Check status of working group", which leads to the exclusive gateway that has two possible results. In case the working group is still active, the event "Send current issue list" happens and the process comes back to the first intermediate timer event. If the gateway result is "No", then the process ends. The model shows that the data object "Issue list" is necessary for the event "Send current issue list" to occur.

1.1.4 Process Discovery

Nowadays, most process models used by companies are not based on real data, but are made by hand from knowledge and experience [25, Chapter 3]. This approach involves several problems, the most important of which is the fact that resulting models do not represent what actually happens. Moreover, these models tend to be influenced by the subjectivity of a person, who may not be perfectly familiar with all the steps of the process. Process mining algorithms for process discovery aim to solve these problems by building a process model from real data present in the event logs.

In order to perform this task, it is not enough to build a model that allows the execution of all the traces contained in the event log, because this would include anomalous paths. Moreover, given the large number of possible variants, the resulting model would be impossible to

understand. We refer to it as *Spaghetti process* (Fig. 1.5). In a process discovery task it is therefore necessary to reach a compromise between *overfitting* and *underfitting*. In the first case, the model is able to generalize most of the traces in the event log, but it often results in an overly complicated model. In the second case, the resulting model is simple and easy to understand, but it is unable to replicate most of the traces.

Process discovery algorithms task is to find a trade-off between the following four quality criteria [25, Chapter 6]:

- *Fitness*: the model should allow for the behavior seen in the event log. A model with a high fitness score is able to replay most of the traces in the log.
- *Precision*: the model should not allow for a behavior that is completely unrelated to what was seen in the event log. Precision score is related to underfitting.
- *Generalization*: the model should generalize the example behavior seen in the event log. Generalization is related to overfitting.
- *Simplicity*: the model should be as simple as possible. It quantifies the complexity of the model.

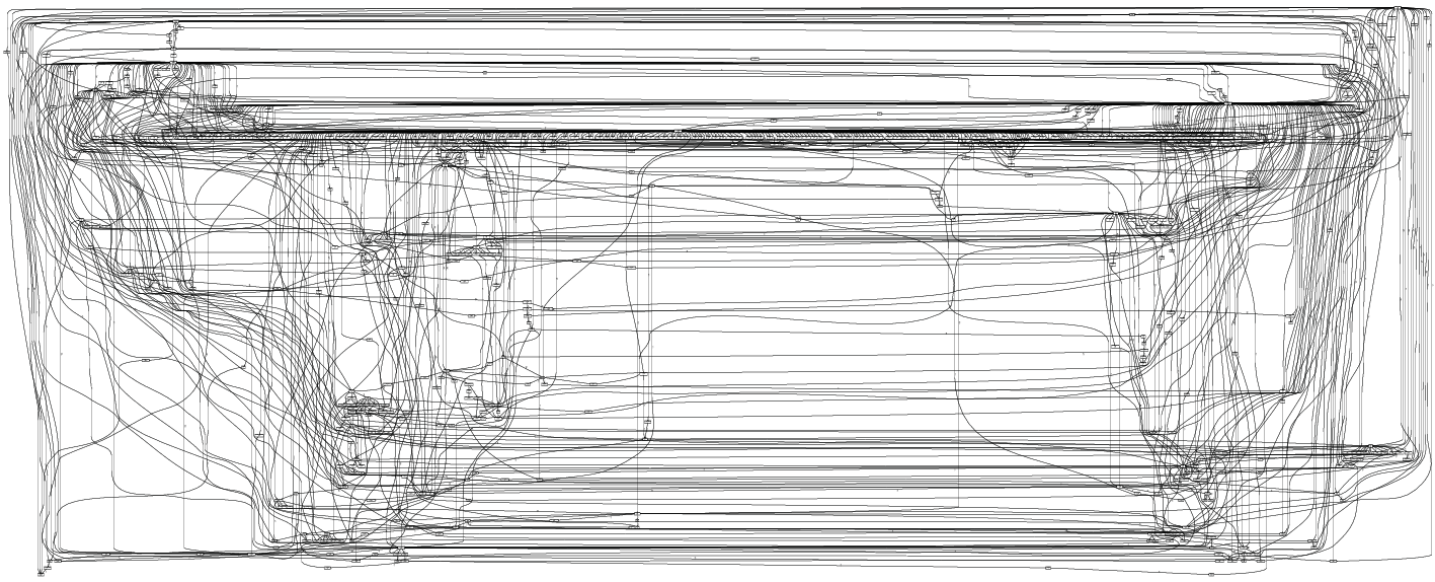

Figure 1.5: Section of a Spaghetti process. When visualizing all the activities and links between activities in a business process, the result is often an overly complicated process.

The task of process discovery can be represented as follows: we assume that there is a set of activities $\mathscr{A}$. The goal is to decide which activities need to be executed, in what order and

which are the conditions that govern them. The two main challenges of process discovery are the presence of noise in the event log, i.e., rare and infrequent behavior, and the lack of negative samples [25, Chapter 6]. The latter refers to the fact that the log only provides example of possible behaviors and we do not have explicit examples of not allowed traces. Maybe a particular path is possible but it just never happens before. Because of this, human knowledge is fundamental in supporting the creation of a process model.

1.1.5 Performance Analysis

Event logs can be used to check the quality of an existing process, for example by quantifying the performance of the current state of operations. Generally, each company defines a set of Key Performance Indicators (KPIs), which are monitoring indicators that assess the performance and the quality of processes. An example of set of typical KPIs that can be used by a company are [34] [35] [36]: amount of waste in production, rework rate, customer satisfaction, lead time standard, net order value, sales growth rate, and efficiency of teamwork.

Companies often rely on external platforms to monitor indicators, such as *Celonis* or *SAP Signavio* [37]. These platforms provide tools for real-time monitoring of KPI values and allow the creation of highly customized dashboards that can contain graphs, messages, and others informative objects. Moreover, they provide the implementation of notification systems that can, for example, send messages to designated users to alert them of anomalies. Some of these platforms are also able to identify the root cause of the anomalies and propose the users solutions for its timely resolution.

1.2 Data Mining

Data mining is the process of searching, discovering and extracting patterns from large raw data sets using data analysis techniques. The extracted information is usually used to improve the processes from which it is derived [38]. One of the most frequent use involves the improvement of customer experience [39]. In this particular case, for example, it is possible to analyze the conversations consumers have with staff or with bots on a website by using NLP techniques that can perform sentimental analysis of reported conversations [40]. Moreover, it is possible to derive data regarding users navigation on the website to find out, for example, that a particular research or form is time-consuming to find or fill out. The results of these analysis can be used to propose improvements. Data mining has similar goals and methods to process mining, but

the data it uses for analysis is not part of a process.

Data mining input is a table, where rows are referred to as *instances* and columns are called *variables*. Variables can be of two types: categorical, such as color or car type, and numerical, which is a continuous value. Since we can not directly use raw data as input for data mining algorithms as they may have nonstandard formatting, data are typically preprocessed. Starting from a process mining dataset, it is possible to perform a *feature extraction*, which refers to the operation of convert an event log into a data mining dataset. Feature extraction is mainly used to perform clustering operations on the data in order to derive additional information about the data being used. For example, it is possible to find that two types of material from different vendors belong to the same cluster and thus are somehow related each others, as they may go through similar processes.

Data mining techniques can be classified in two main categories: *supervised learning* and *unsupervised learning*. Supervised learning assumes the data to be labeled and the goal is to find a function that can label each data point according to some rule, by minimizing a defined error function. Supervised learning techniques can be divided in classification and regression. In a classification task, the label can assume a set of predefined values, whereas in a regression task the label is a numerical value. Unsupervised learning assumes the data not to be labeled and it aims to divide data into similar group according to some functions. The most used supervised learning algorithms are [41] [42]: k-Nearest Neighbors, Linear Regression, Logistic Regression, Support Vector Machine (SVM), Decision Trees, and Neural Networks. On the other hand, unsupervised learning cover field such as Clustering [43] (K-Means, DBSCAN), Anomaly Detection [44] (Isolation Forest), and Dimensionality reduction [45] (Principal Component Analysis, t-Distributed Stochastic Neighbor Embedding).

1.2.1 Metrics of a classification problem

Given a supervised machine learning problem, we want to evaluate the performance of a classifier. This problem is nontrivial since it does not exist a universal metric that can be used [46]. Let's consider, for example, the confusion matrix of a binary classification problem, as shown in Fig. 1.6. The diagonal of the matrix represents the correct predictions: True Positive (TP) are the positive label predicted as positive and True Negative (TN) are the negative label predicted as such. The other two elements represent the incorrect classification: False Positive (FP) represents the outcomes predicted as positive but that are negative in reality, and the False Negative (FN) are the outcomes predicted as negative incorrectly. The confusion matrix can show the

number of the predicted outcomes, the ratio computed over the whole number of samples, the ratio computed over true values, or the ratio computed over predicted labels, depending on the chosen normalization. A good classifier has a high number of elements over the diagonal and few in other positions.

		Predicted values: Positive	Predicted values: Negative
True values	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Figure 1.6: Confusion matrix for a binary classification problem. Rows refer to the true label, whereas columns refer to the predicted one. There exists four possible outcomes.

Which metric to use to evaluate the performance depends on the problem we are investigating [47]. Sometimes we are interested in achieving the highest possible accuracy, whereas in other occasions we are more interested in the accuracy of the positive labels. In some problems it is important to punish misclassifications of only one label and not both [48], for example in the case of a test for a disease. In this case, in fact, it is important that the test accurately detects diseased subjects but it is not important if it detects false positives, since it is enough to run the test a second time to reveal the error. Another key aspect to consider is the class balance, as some metrics may return a high value when in fact the classifier is failing in the prediction task. The most commonly used classification metrics are shown below [49].

Accuracy is the most widely used metric in classification problems. It can be computed as follows:

$$accuracy = \frac{TP + TN}{TP + FP + FN + TN}. \tag{1.1}$$

It represents the ratio of correct classification made by the predictor. The main limitation concerns imbalanced data, since in this case we can achieve a high accuracy score but one label can be total misclassified.

Another popular metric is *precision*, given by

$$precision = \frac{TP}{TP + FP}, \tag{1.2}$$

which quantify the ratio of correct classification over the positive predicted samples. This metric quantify the rate of positive prediction that are actually positive. Precision is used to quan-

tify a classifier where the number of False Positive must remains low.

A similar metric to precision is the *recall*, defined as:

$$recall = \frac{TP}{TP + FN}, \tag{1.3}$$

which differs from precision because the ratio is computed over the positive true samples. Recall quantifies how well the classifier is able to identify actual positive. For example, we have a facial-recognition device that has the task to detect criminals. It is very important that this device is able to recognize any criminal that he sees, and therefore has a high score of True Positive and a low score of False Negative. In this example, recall can be a first metric to be used since it is not influenced by the amount of False Positive.

Since precision and recall are similar but computed over different total number of samples, a trade off between these two can represent an appropriate metric for the evaluation of the performance of a classifier. For this reason, we compute the *F1 score*, which is the harmonic mean between precision and recall, computed as follows:

$$F1 = \frac{2 \times precision \times recall}{precision + recall}. \tag{1.4}$$

When approaching a binary classification task with imbalance class and where all labels have equal importance, previous scores can lead to inappropriate results [50]. The *Geometric Mean score* (G-mean) can be used to overcome the problem. If one class is total misleading, the G-Mean score is equal to 0. It is evaluated using:

$$G - Mean = \sqrt{specificity \times recall}, \tag{1.5}$$

where *specificity* is given by:

$$specificity = \frac{TN}{TN + FP}. \tag{1.6}$$

Another important metrics is the *AUC-ROC*, which stands for Area Under the Curve (AUC) Receiver Operator Characteristic (ROC). ROC is the plot of the cumulative distribution function of the TP against the FP at various threshold values. An example of the ROC curve can be seen in Fig. 1.7. The AUC is the area under the ROC curve. An AUC value near 1 represent a good classifier, whereas a value near 0.5 is the value of a random classifier. For a multiclass

classification problem, the AUC-ROC curve is usually evaluated as a weighted mean of the AUC-ROC for each label.

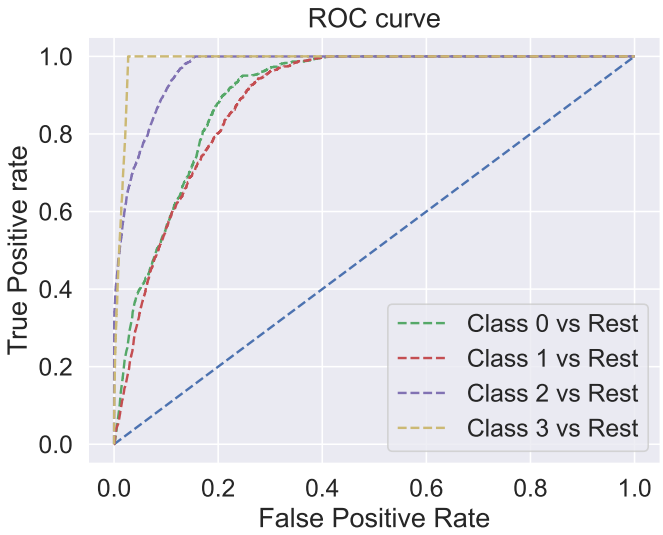

Figure 1.7: Receiver Operator Characteristic (ROC) curve for a Multi-Class Classification. For each predicted label, the ROC is computer as "Class vs the Rest". For the evaluation of the classifier, usually the weighted mean of all AUC for each class is computed.

Probability Metrics

In a classification problem, sometimes we are interested in evaluating the probability estimation accuracy performance, instead of the classified label. In this case, we can evaluate the performance using probability metrics, also known as *scoring rules*, which quantify how the predicted probability distribution matches the known probability distribution. The most used scoring rules are the *log-loss* and the *Brier score*.

The log-loss (*cross-entropy loss* for a multiclass classification problems) is a *local* scoring rule, meaning that the level of accordance must be determined only by what is actually observed [51]. The cross-entropy loss of the predicted probability distribution $\hat{\mathbf{y}}$, given the true probability distribution $\mathbf{y}$, is the following:

$$L(\hat{\mathbf{y}}, \mathbf{y}) = -\frac{1}{N}\sum_{i=0}^{N}\sum_{j=0}^{R} y_{ij}\log(o_{ij}). \tag{1.7}$$

In this equation, N is the total number of predicted samples, R is the total number of classes,

o_{ij} is the predicted probability of digit j for sample i and y_{ij} is the true probability of the class j for sample i. When the target are one-hot encoded label, the only non-zero term is the one of the true label. All others prediction probabilities have zero weight in the loss evaluation.

The log-loss will explode if we observe an event that have 0 true probability, meaning an impossible event. This problem is overcome by the Brier score, which measures the mean squared difference between the predicted probability distribution and the true probability distribution [52]. The Brier score is given by:

$$BS(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{N}\sum_{i=0}^{N}\sum_{j=0}^{R}(o_{ij} - y_{ij})^2. \tag{1.8}$$

The choice of the probability metric depends on the type of error one want to consider. The log-loss metric is concentrated on the difference between the true and the predicted probability, whereas the Brier score takes into account also the probability distribution of the rest of the labels.

2

Predictive Process Monitoring

Predictive process monitoring is a technique that combines process mining and predictive analytics to forecast process attributes, such as the next activity or the remaining time of a process. Firstly, we exploit process mining techniques to analyze historical process data in order to understand the actual process flow, identify patterns, and extract relevant information [53]. This analysis provides insights into the process behavior and it can uncover inefficiencies, bottlenecks, and compliance violations. Once understood the process course, we apply predictive analytics techniques to build models trained using historical process data.

Companies can use the predictive models to forecast process outcomes, estimate process performances, detect anomalies, and predict the likelihood of certain events occurring in the future [54] [55]. This enable organizations to proactively identify and prevent process issues or deviations [56] [57]. By leveraging predictive insights, businesses can optimize their processes and improve operational efficiency.

2.1 Literature Review

Several works exist in literature about the predictive process monitoring task. However, researchers use very different techniques and evaluation datasets, making the comparison and the generation of a baseline approach challenging. In the work by Rama-Maneiro et al. [58], the authors provide a systematic literature review of the most used deep learning techniques. In their work, 10 different approaches are tested over 12 publicly available process logs, evaluat-

ing the results on different prediction tasks, such as *next activity*, *activity suffix*, and *remaining time*. Outcomes show that approaches that achieve the highest scores are the ones proposed by Hinkka et al. [59], Pasquadibisceglie et al. [53], Tax et al. [60], and Camargo et al. [61]. Results highlight that overall using attributes as inputs to the models may not benefit the predictive task, suggesting to incrementally add the attributes and observe their effects. Moreover, hyper-parameters optimization does not appear to have a significant impact in model scores. Finally, approaches that perform a direct prediction of the remaining time achieve better results than approaches that sum up the next activity time until the last activity. Regarding the neural network architecture, Recurrent Neural Networks (RNNs) outperform Convolutional Neural Networks (CNNs), even if the latter are usually faster in training.

The systematic literature review done by Neu et al. [62] focuses the attention towards encoding methods and neural network architectures. The work highlights that it does not exist a universal approach to encode process mining data. For instance, it is possible to process categorical data using either one-hot encoding or via embedding methods. When the number of categories is large (more than 20), the latter leads to better results. The study further distinguishes regression tasks, primarily concerned with time-based prediction, from classification tasks, which revolve around activities or event features. Results indicate that incorporating more input features generally leads to improved performances. Moreover, the review emphasizes the importance of addressing the issue of imbalanced classes, suggesting that further investigations are necessary to better understand and tackle this problem.

One of the most used neural network type for predictive process monitoring task is the Long-Short Time Memory (LSTM) neural network. In the work conducted by Tax et al. [60], LSTM networks achieve better results than previous experiments, establishing the proposed architecture as a starting point for future researches. The study focuses on the evaluation of suffix traces and the remaining time of a trace by continuously iterating the next event prediction. The network inputs are one-hot encoded activities and three temporal information variables, representing the time difference from the previous activity, the time of the day and the day of the week when the activity is performed. Three different architecture setups are tested: the first one is specific to each prediction task, and the other two involve multi-task learning approaches. Camargo et al. [61] provide a similar research, comparing three different LSTM neural network architectures for two tasks: reproducing an event-log and predicting outcomes. Unlike previous works, this study uses categorical encoding for processing inputs such as the activity, the resource performing the activity, and the timestamp of the activity. The three architectures differ in how information is shared across these categories. The first one is *specialized*, i.e. LSTM

layers are not shared across the three main channels. The second architecture is *categorical shared*, i.e. the activity and resource information is shared across LSTM layers but not the time information. Finally, the third one is *fully shared*. Both studies by Camargo et al. and Tax et al. highlight that the shared model outperforms the specific model when evaluated using Damerau–Levenshtein and MAE metrics.

The work by Navarin et al. [55] tackles the task of predicting the remaining time of the process using LSTM neural networks, achieving better performance than [60]. In comparison to the previous study, the model proposed is able to deal with repeated activities, can take into account event attributes, and has lower training times.

Convolutional Neural Networks' (CNNs) ability to capture spatial dependencies in data can be exploited to tackle the predictive task. By mapping a categorical variable to a vector representation, CNNs are able to capture meaningful relationships. The work conducted by Di Mauro et al. [63] explores the usage of 1-dimensional CNN architectures in the prediction task, by investigating the possibility of CNNs to outperform RNN architectures. Authors encode activities using embedding layers. Results suggest that 1-dimensional CNN architectures have longer effective memory and can achieve comparable or superior performance to traditional RNN architectures, while being computationally more efficient.

The research carried out by Pasquadibisceglie et al. [53] provides another investigation on the usage of CNNs in predicting scenarios. They focus on exploring a novel process data engineering scheme that transforms temporal data into spatial representations. This approach opens up new possibilities for leveraging CNNs' strengths in capturing spatial dependencies. However, one limitation highlighted in the study pertains the prediction of non-frequent activities. To address this limitation, the work suggests to employ oversampling techniques. These can involve, for example, the artificial increase of the representation of undersampled activities in the training data. By balancing the data distribution, it is possible to improve the predictive performance for non-frequent activities.

Metzger et al. [64] highlight two important limitations when using RNN architectures for prediction tasks. Firstly, they address the issue of *cycles* in activities, which refer to the repetition of a portion of the process. Traditional forecast methods are unable to predict activity cycles, especially when predicting the trace suffix using an iterative approach. Moreover, this produces an accumulation of prediction errors. To overcome this limitation, the proposed solution focuses on directly predicting the final process outcome, which leads to better results. Secondly, the study highlights the challenge posed by parallel activities in the prediction task. Authors propose a solution that encodes parallel branches as an activity attribute. However, this ap-

proach only results in a marginal improvement. Furthermore, in order to address the problem of imbalanced classes, the paper suggests the usage of the Matthews Correlation Coefficient (MCC) as an accuracy score. The MCC better reflects the real-life complexity of imbalanced class distributions and provides a more comprehensive evaluation of prediction performance.

Given the large number of event attributes in business event-log, the work of Hinkka et al. [59] explores a clustering technique that can be used to take advantage of any additional information. The proposed approach offers a trade-off between prediction accuracy and computational time. The user can define the maximum number of clusters, and thus the length of the resulting one-hot encoding. The input vector used embeds the cluster labels, along with the event and its attributes. Authors test the approach on five different datasets. Results achieved reveal that in four out of five experiments, applying the clustering technique to the attributes input vectors produces superior results compared to encoding event attribute values alone.

Evermann et al. [28] propose a predictive approach inspired by NLP, given the similarity between the prediction targets. This study treats the activity sequences as a word and generates the neural network inputs by encoding the attributes through an embedding space. One unique aspect of this approach is that the temporal information is incorporated using fixed time steps of 1 minute. For example, if activity A lasts for 3 minutes, then it is encoded as AAA in the sequence representation.

Some relevant anomaly detection studies on business process dataset are presented by Lahan et al. [65] and Nolle et al. [66]. In the former, authors employ a LSTM neural network for a two-step research. In the first step, authors train a LSTM model on the prediction task of the next activity in a sequence. In the second step, they use the trained model to predict the next activity for each window of fixed length in the traces. After predicting all the activities in a trace, authors assign an anomaly score to each predicted event. If the trace has at least one anomaly score above a certain threshold, then the trace is considered anomalous. The study evaluates the performance of the methods using the F1 score, that provides an assessment of the algorithm's ability to detect anomalies effectively.

Nolle et al. [66] conduct a similar work using autoencoder architectures. In the study, authors encode activities using one-hot encoding, and train the autoencoder on the entire set of traces. Then, they use the network to reproduce the traces. By measuring the Mean Squared Error between the reproduced traces and the real trace, it is possible to identify anomalous traces and the activity that causes the anomaly.

2.2 Deep Learning techniques for prediction task

In recent years, there has been a widespread application of deep learning and neural network techniques in prediction tasks. Among these, the most used are Recurrent Neural Networks and their derivatives, such as Long Short-Term Memory and Gated Recurrent Units networks, as well as Convolutional Neural Networks. We can train these models on different types of inputs and outputs. This results in a multitude of approaches that can be used to tackle predictive tasks. Some examples of the most frequently used models are [67, Chapter 15]:

- Sequence-to-Sequence (*seq2seq*): this model takes a sequence as input and produces a sequence as output. We use this model to predict values at each time step.
- Sequence-to-Vector (*seq2vec*): this model takes a sequence as input and produce a vector as output. We use this model to forecast several steps ahead simultaneously. It can be used as an *encoder*. In this case it transforms the input into a fixed-length vector called the *context vector* or *hidden state*.
- Vector-to-Sequence (*vec2sec*): this model takes the same vector multiple times as input and produces a sequence as output. It can be used as a *decoder*. In this case, it takes the context vector as the initial hidden state and generates the output sequence step by step. At each time step, the decoder uses the previous generated output as input to predict the next step in the sequence.
- Autoencoder: this model takes a sequence as input and produces a sequence as output. In this case, the input sequence is transformed in a lower-dimensional vector by an encoder, and then again in a higher-dimensional sequence by a decoder.

Different types of forecast require different models. For instance, in next activity prediction tasks, we can use either a *seq2vec* or *seq2seq* model. The former provides an immediate forecast, whereas the latter make a prediction for each time step. In this case, the next activity we consider is the last one. For trace suffix prediction, we can use a *seq2vec* model to predict the next activity iteratively [60]. Alternatively, we can train a *seq2seq* model to predict next values at each time step. This approach offers the advantage of incorporating error gradients from multiple time steps, enhancing training stability and speed [67, Chapter 15].

Sequence-to-Sequence models and autoencoders are typically used in language generation tasks [68], chatbot applications [69], translation tasks [70], and speech recognition [71]. In these scenarios, a text is provided as an input sequence and the model generates a corresponding output text that could be, for instance, a translated version of the original text or an answer

to a question. On the other hand, a *seq2vec* model can be employed for dimensionality reduction [72], sequence embeddings [73], sentiment analysis, feature classification [74], and the prediction of the next word in a sentence.

2.2.1 Recurrent Neural Networks

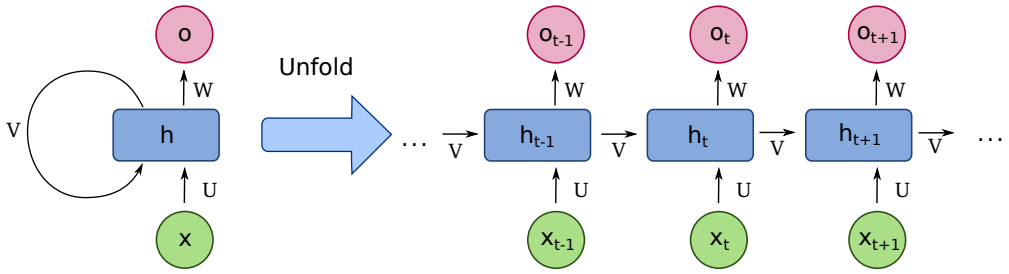

Figure 2.1: Recurrent neural networks unfold over time. The output of the hidden node serves as input for the same node in the subsequent time steps. In this way the network creates a temporal information that is shared across all time steps.

Recurrent Neural Networks are ones of the most widely used types of neural network in problems involving sequential data. RNNs application varies from language translation, speech recognition, stock price prediction, and activity prediction [75]. The main feature of RNNs is their ability to utilize previous outputs as inputs to the same nodes, making them able to capture temporal information from preceding steps. Fig. 2.1 shows a RNN unit unfold over time.

The *state* of the node, represented as $\mathbf{h}_t$, and the *output*, represented as $\mathbf{o}_t$, at time step t, are computed using the following equations:

$$\begin{cases} \mathbf{h}_t = \varphi(\mathbf{U}^\intercal \mathbf{x}_t + \mathbf{V}^\intercal \mathbf{h}_{t-1} + \mathbf{b}) \\ \mathbf{o}_t = \mathbf{W}^\intercal \mathbf{h}_t + \mathbf{c} \\ \hat{\mathbf{y}}_t = \phi(\mathbf{o}_t), \end{cases} \tag{2.1}$$

where $\mathbf{x}_t$ is the *input vector* at the current time step t; $\mathbf{h}_{t-1}$ is the *state vector* of the previous time step $t-1$; $\mathbf{o}_t$ is the *output vector* at time step t; $\hat{\mathbf{y}}_t$ is the *output* of the network at time step t, that is simply $\mathbf{o}_t$ transformed using an activation function that depends on the problem nature. The other terms $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$ are the weight matrices of the terms, whereas $\mathbf{b}$ and $\mathbf{c}$ are the *bias*

vectors. φ is the state activation function, which is usually the *tanh*; ϕ is the activation function of the last layer, which for example can be a *softmax* for a probabilistic interpretation.

In a RNN, each state of the previous time steps influences the future ones. Essentially, a neuron generates an output, and then feeds the output back to itself. For example, we consider the sequence $\langle A, B, C \rangle$. The aim of the problem is to find the probability of the next activity in the sequence, that can be activity D, E or F.

1. At time step $t = 0$, the node receives the first activity A as input encoded in $\mathbf{x}_0$. The state of the node changes to $\mathbf{h}_0$ and the node produces an output $\mathbf{o}_0$.
2. At the second time step, the node takes the activity B as a vector $\mathbf{x}_1$ and combines it with the previous state $\mathbf{h}_0$, which contains information about activity A. Then it generates an output $\mathbf{o}_1$.
3. Similarly, at the third time step, the node takes input $\mathbf{x}_2$ and $\mathbf{h}_1$, producing an output $\mathbf{o}_2$ that encapsulates information about the entire sequence.
4. Finally, the network transforms the last output $\mathbf{o}_2$ into a probabilistic vector using the *softmax* activation function, producing $\hat{\mathbf{y}}_2$, which is a vector representing the probability of each possible activity happening at the next time step.

Regarding the outputs $\mathbf{o}_t$, we are only interested in the last one. The intermediate steps only serve the purpose of updating the state $\mathbf{h}_t$.

In recurrent neural networks, we perform the weights updating using a technique called backpropagation through time (BPTT) [76]. Backpropagation is commonly used in supervised learning problems to minimize the error of a neural network by updating the weights of its nodes. In standard backpropagation, we propagate the input through the network and then we evaluate the error between the network output and the expected output according to a chosen loss function. Then we compute the derivatives of the weights with respect to the loss function and update the weights in a way that minimizes the error. BPTT works similarly to standard backpropagation, but in addition it has the complexity of dependencies across previous time steps. As a result, we need to differentiate the error function with respect to the previous state, allowing the gradients to flow backward across time steps. To facilitate this, we unroll the network over time and we apply regular backpropagation. In this context, $\mathbf{o}_t$ represents the output of a cell at a specific time step, while $\hat{\mathbf{y}}_t$ represents the output of the entire network at time step t. Given the loss function:

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{t=1}^{T} \mathcal{L}_t(\hat{\mathbf{y}}_t, \mathbf{y}_t), \tag{2.2}$$

the backpropagation through time updates the weights as follows:

$$\begin{cases} \dfrac{\partial \mathcal{L}}{\partial \mathbf{W}} = \displaystyle\sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial \mathbf{W}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{W}} \\ \dfrac{\partial \mathcal{L}}{\partial \mathbf{V}} = \displaystyle\sum_{t=1}^{T} \sum_{k=1}^{t+1} \frac{\partial \mathcal{L}_{t+1}}{\partial \hat{\mathbf{y}}_{t+1}} \frac{\partial \hat{\mathbf{y}}_{t+1}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{V}} \\ \dfrac{\partial \mathcal{L}}{\partial \mathbf{U}} = \displaystyle\sum_{t=1}^{T} \sum_{k=1}^{t+1} \frac{\partial \mathcal{L}_{t+1}}{\partial \hat{\mathbf{y}}_{t+1}} \frac{\partial \hat{\mathbf{y}}_{t+1}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{U}}. \end{cases} \tag{2.3}$$

RNNs face two major limitations: the vanishing and exploding gradient [77], and the short-term memory problems. The former arises due to the BPTT algorithm, which can result in deep and unrolled RNNs, that lead to unstable gradients. When gradients become extremely small, weights are not updated effectively, preventing the network from reaching the minimum (vanishing gradient). Whereas, when gradients are too large, weights updates become excessively large, causing the gradient to diverge (exploding gradient). The short-term memory problem occurs when RNNs are trained on long sequences. In such cases, the network tends to forget information from earlier inputs in the sequence as it progresses. This limitation threatens the RNN's ability to capture long-term dependencies. To address these issues, variants of RNNs have been developed, including Long Short-Term Memory (LSTM) [78] and Gated Recurrent Units (GRU) neural networks [79].

2.2.2 Long Short-Term Memory

Long Short-Term Memory (LSTM) neural networks are a type of RNN that effectively addresses the short-term memory problem. They accomplish this by incorporating a *cell state*, denoted as $\mathbf{c}_t$, which enables the learning of long-term dependencies. Unlike traditional RNNs that rely solely on the previous state $\mathbf{h}_{t-1}$, LSTMs employ a more complex architecture with gated structures. These allow to control the flow of information in and out of the cell state, using the *sigmoid* activation function [78]. An output close to 0 means that the information is disregarded, while an output close to 1 means that the information is preserved. A detailed view of a LSTM unit can be seen in Fig. 2.2. Its input consists of the current time step input $\mathbf{x}_t$, the hidden state $\mathbf{h}_{t-1}$, and the cell state $\mathbf{c}_{t-1}$ from the previous time step. In comparison, a standard recurrent cell misses the cell state component.

The operation of a LSTM cell can be described step-by-step as follows [67, Chapter 15]:

1. In the first step, the input $\mathbf{x}_t$ and the state of the previous time step $\mathbf{h}_{t-1}$ pass through the

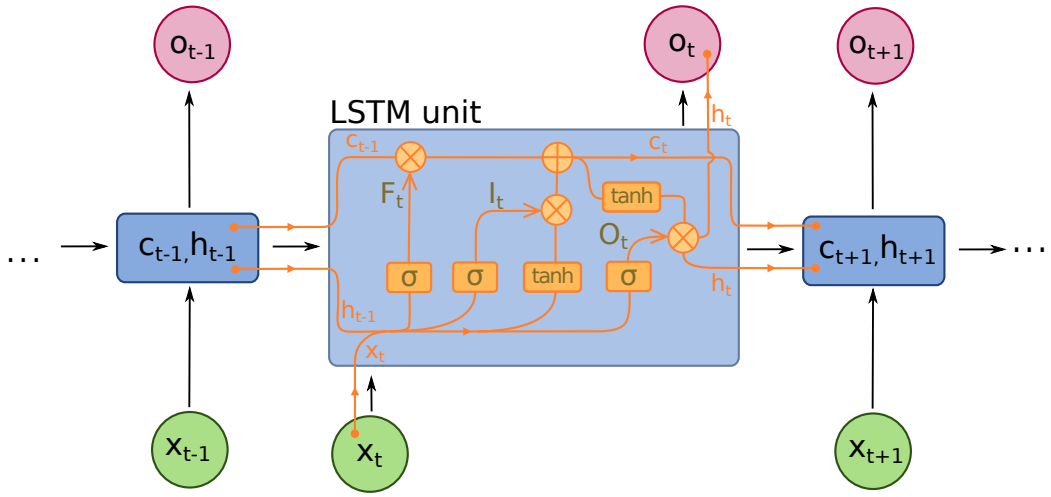

Figure 2.2: Long Short-Term Memory unit in detail and unrolled. We show the composition of a LSTM unit, where we can identify the various gates and their activation functions used to select or discard information.

forget gate, that produces the output $\mathbf{f}_t$. Here the gate selects which information from the cell state to keep and which to throw away.

2. In parallel, the input $\mathbf{x}_t$ and the state of the previous time step $\mathbf{h}_{t-1}$ are used to compute $\mathbf{i}_t$ and $\mathbf{g}_t$. Later these values pass through the *input gate*, which decides which values are updated and which values can be added to the cell state.

3. Then, the unit updates the cell state $\mathbf{c}_t$ accordingly to the previous decisions.

4. Finally, the *output gate* takes as input $\mathbf{o}_t$, which is the output of the layer, and the current cell state $\mathbf{c}_t$. Then, it decides which information should be read and output at this time step, producing $\hat{\mathbf{y}}_t = \mathbf{h}_t$.

The equations that govern the process are the following:

$$\begin{cases} \mathbf{f}_t = \sigma(\mathbf{W}_{xf}^\intercal \mathbf{x}_t + \mathbf{W}_{hf}^\intercal \mathbf{h}_{t-1} + \mathbf{b}_f) \\ \mathbf{i}_t = \sigma(\mathbf{W}_{xi}^\intercal \mathbf{x}_t + \mathbf{W}_{hi}^\intercal \mathbf{h}_{t-1} + \mathbf{b}_i) \\ \mathbf{o}_t = \sigma(\mathbf{W}_{xo}^\intercal \mathbf{x}_t + \mathbf{W}_{ho}^\intercal \mathbf{h}_{t-1} + \mathbf{b}_o) \\ \mathbf{g}_t = \tanh(\mathbf{W}_{xg}^\intercal \mathbf{x}_t + \mathbf{W}_{hg}^\intercal \mathbf{h}_{t-1} + \mathbf{b}_g) \\ \mathbf{c}_t = \mathbf{f}_t \otimes \mathbf{c}_{t-1} + \mathbf{i}_t \otimes \mathbf{g}_t \\ \hat{\mathbf{y}}_t = \mathbf{h}_t = \mathbf{o}_t \otimes \tanh(\mathbf{c}_t), \end{cases} \tag{2.4}$$

where the matrices $\mathbf{W}_{x\cdot}$ and $\mathbf{W}_{h\cdot}$ are the weight matrices of each of the four layers f, i, o, and g, for their connection to the input vector $\mathbf{x}_t$ and to the previous short-term state $\mathbf{h}_{t-1}$, respectively.

LSTM cells have the ability to identify and retain crucial information by storing it within the long-term memory state and retrieving it when necessary [80]. This characteristic explains the success of LSTM networks in capturing long-term patterns within various types of data, including time series, texts, audio recordings. LSTMs contribute to performance improvement in tasks such as language modeling [81], speech recognition [82], machine translation [83], and sentiment analysis [84].

2.2.3 Gated Recurrent Unit

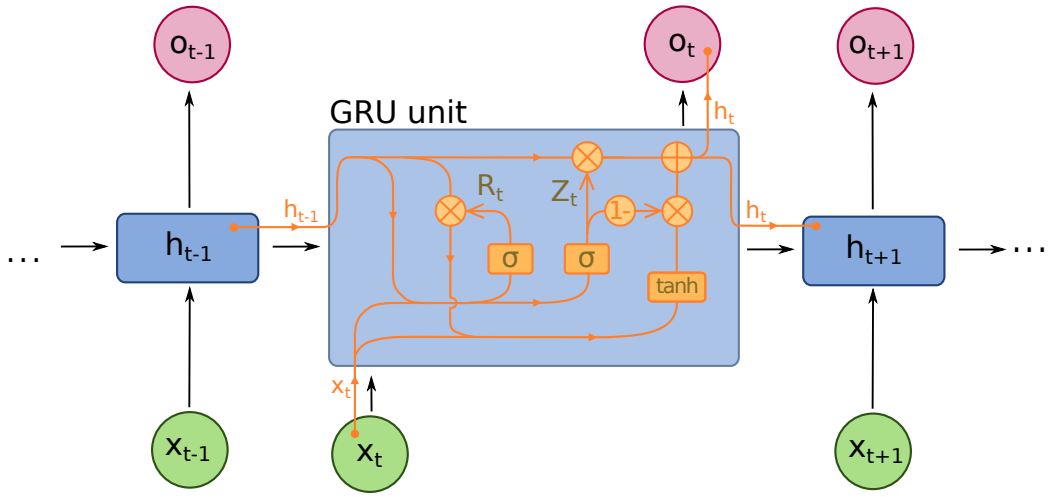

Figure 2.3: Gated Recurrent Unit in details and unrolled. Inside the unit, we can identify the various gates and their activation functions used to select or discard information. In comparison with the LSTM unit, the GRU unit has less gates and it does not have the *cell state*.

The Gated Recurrent Unit (GRU) is a simplified variant of the LSTM cell that replaces the three gates (input gate, forget gate, and output gate) with just two gates: the *reset gate* and the *update gate* [79]. The reset gate controls how much of the past information contained in the previous output vector $\mathbf{h}_{t-1}$ is disregarded or forgotten. On the other hand, the update gate determines how much of the new information should be incorporated into the current hidden state. Fig. 2.3 shows a GRU unit in details.

The operation of a GRU cell can be described as follows [67, Chapter 15]:

1. In the first step, the *reset gate* takes as input the previous output $\mathbf{h}_{t-1}$ and the current

input $\mathbf{x}_t$ and produces as output $\mathbf{r}_t$. The gate selects how much of the past information to forget.

2. Meanwhile, the same inputs are passed to the *update gate*, that produces as output $\mathbf{z}_t$. The gate selects how much of the new information should be used to update the current state.

3. Then, the cell uses the output of the reset gate to compute the *candidate hidden state* $\hat{\mathbf{h}}_t$, which is the previous hidden state minus the blocked information.

4. Finally, the output of the cell $\hat{\mathbf{y}}_t = \mathbf{h}_t$ is computed as a trade-off between the current memory content $\hat{\mathbf{h}}_t$ and the previous output $\mathbf{h}_{t-1}$.

The equations that govern the process are the following:

$$\begin{cases} \mathbf{z}_t = \sigma(\mathbf{W}_{xz}^\intercal \mathbf{x}_t + \mathbf{W}_{hz}^\intercal \mathbf{h}_{t-1} + \mathbf{b}_z) \\ \mathbf{r}_t = \sigma(\mathbf{W}_{xr}^\intercal \mathbf{x}_t + \mathbf{W}_{hr}^\intercal \mathbf{h}_{t-1} + \mathbf{b}_r) \\ \hat{\mathbf{h}}_t = \tanh(\mathbf{W}_{xh}^\intercal \mathbf{x}_t + \mathbf{W}_{hh}^\intercal (\mathbf{r}_t \otimes \mathbf{h}_{t-1}) + \mathbf{b}_h) \\ \hat{\mathbf{y}}_t = \mathbf{h}_t = \mathbf{z}_t \otimes \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \otimes \hat{\mathbf{h}}_t, \end{cases} \tag{2.5}$$

where the matrices $\mathbf{W}_{x\cdot}$ and $\mathbf{W}_{h\cdot}$ are the weight matrices of each of the three layers z, r, and h, for their connection to the input vector $\mathbf{x}_t$ and to the previous output vector $\mathbf{h}_{t-1}$, respectively.

GRU-based networks are capable of capturing dependencies over longer sequences while being computationally less complex than LSTM cells [85]. GRUs have been widely used and have demonstrated strong performance in tasks involving sequential data analysis, such as natural language processing [86] and speech recognition [87].

While it may not completely eliminate the problem, the GRU architecture helps to overcome the vanishing gradient issue compared to traditional RNNs [77][88]. Its gates allow the cell to reset or forget past information, while controlling the flow of new information into the hidden state.

2.2.4 Autoencoder

Autoencoder is a different type of neural network architecture that can be used in unsupervised and semi-supervised learning with temporal data. It is capable of learning a hidden representation of the data by framing the unsupervised task as a supervised one, where the network input is used as target. In this way, the network is in charge of reconstructing the input.

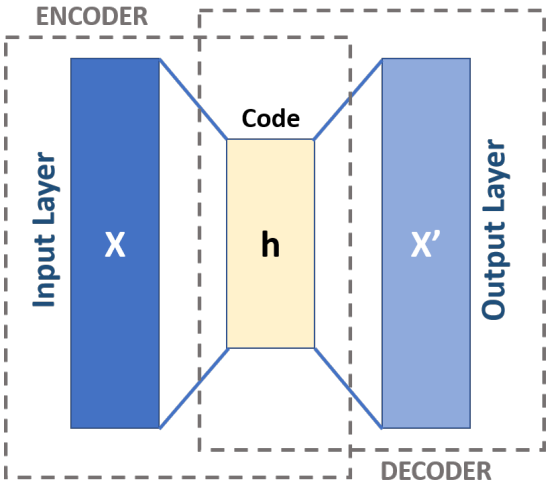

Figure 2.4: Autoencoder architecture. The first half represents the encoder, which projects the input into a lower-dimensional representation, encoded in the bottleneck layer. The second half of the architecture is the decoder, which projects the lower-dimensional representation of the bottleneck layer into the original dimension of the input. An autoencoder aims to reconstruct the input.

Autoencoders consist of two main parts: the *encoder* and the *decoder*. A schematic representation is shown in Fig. 2.4. The encoder learns a lower-dimensional representation of the input, while the decoder recovers the original input. The power of autoencoders lies in the bottleneck layer, which is the lower-dimensional hidden layer and the last layer of the encoder. The number of nodes in this layer determines the encoding dimension of the input. A common use of autoencoders is as a dimensionality reduction tool [89] [90]. After training the autoencoder, the bottleneck layer can be used as input in unsupervised or supervised problems. In this way, autoencoders are capable of learning non-linear hidden correlations, unlike other dimensionality reduction methods that rely on linear projections.

Autoencoders have various applications, including inputs reconstruction and denoising [91] [92] [93]. Moreover, they can be used for the purpose of anomaly detection [94] [95] [96]. In this case, the network is trained to learn its reconstruction patterns. When an anomaly input is passed to the network, the resulting reconstruction error is be high, indicating that the sample is anomalous. The problem can be mapped in a semi-supervised learning task by setting a certain threshold that defines which samples are considered anomalous, or by selecting only anomaly-free samples as inputs.

2.3 Process Forecasting

Neural network architectures previously described can be used to forecast process outcomes. In this context, a wide variety of targets exists, which can be divided mainly into classification tasks or regression tasks [58]. Regarding the former category, the most common predicted features include next activity [53] [59] [61] [63] [28][65] , activity suffix [60] [61] [65] [66], event and case attributes [97]. On the other hand, the main important regression targets are: timestamp of next activity [60] [98], remaining time of a trace [55] [60] [61], and duration of activities [28].

Future prediction is usually framed to a supervised learning framework. In next activity prediction tasks, the output of the neural network is a probability vector associated with each possible outcome. The next activity is simply taken by applying the *argmax* function. However, this approach may not be optimal for trace suffix problems as it often fails to consider anomalies, leading to potential loops of activities. In this case, employing *random sampling* instead of *argmax* leads to a higher and more accurate performance, as it reproduces the actual trace more frequently [28] [61].

Studies use different metrics to evaluate the performance of the models. In the next activity prediction task, the most used metric is *accuracy* [53] [61], that provides a measure of the ability of model to predict the true label. On the other hand, regarding the activity suffix task, many studies rely on the Damerau-Levenshtein distance metric [28] [58] [60]. This metric takes into account the parallel nature of activities and quantifies the minimum number of insertions, deletions, substitutions, and transpositions (swapping of adjacent symbols) required to transform one string into another. It provides a comprehensive measure of the dissimilarity between the predicted suffix and the actual suffix.

Concerning time-based prediction tasks, researches prefer the Mean Absolute Error (MAE) over alternatives such as Mean Squared Error (MSE) [55] [60] [61], since the latter tends to amplify the impact of outliers, whereas the former provides a more direct measure of the average prediction error.

3

Dataset

Process mining datasets typically consist in event logs that capture the sequences of activities, their timestamps, and other relevant information related to real-world processes. While almost every company stores data about internal processes, the majority of this information is considered confidential and can not be made publicly accessible. Therefore, only a limited amount of process mining datasets are available online and most of the researches in the field is conducted using the same ones.

For the purpose of this work, we create a *Procure-to-Pay* (*Purchase-to-Pay*) dataset from scratch. We use the dataset for the production of a process mining demo within the Celonis platform. Celonis provides various tools for data analysis, including process discovery, conformance and checking algorithms. In addition, the platform provides a Python environment that allows the implementation of machine learning and deep learning techniques, as well as preprocessing or data cleansing scripts.

As a starting point, we consider the SAP table format, as it is the most widely used system in the business world. We randomly simulate information contained in some specific SAP tables fields. Every name, number, date, and information in the dataset is fictional and created using random generators. Subsequently, we relate the tables to each other using foreign keys and we generate the data model in order to create the event-log.

3.1 Procure-to-Pay process

Procure-to-Pay, also known as Purchase-to-Pay (P2P), is a series of steps that describe the end-to-end business process of requesting, purchasing, receiving, and paying for goods from external suppliers [99]. The process involves two entities: the buyer, which is the company holding the P2P process, and the vendors, which are the companies selling the products or services required by the buyer. The process can be summarized as follows [100]:

1. Needs identification: the buyer identifies goods or services it wants to buy. This demand can arise from various sources, such as operational requirements, project demands or maintenance needs.
2. Creating *Purchase Requisition* (PR): the buyer creates a requisition for the goods or services it needs. This internal process includes comparing pricing and terms from different vendors and consulting the purchase history. Purchase requisitions usually include price, quantity, and any other relevant information. Requests are reviewed and approved by appropriate authorities within the organization.
3. Sending purchase request: once the purchase is approved internally, the buyer sends a purchase request to the vendor. This request includes details of the desired goods or services, such as quantity or material type.
4. Shipping and invoicing: the vendor ships goods to the buyer and sends an invoice for payment. Invoices typically include agreed-upon price and any applicable terms.
5. Payment: the buyer processes the invoice and makes the payment to the vendor according to agreed-upon terms.
6. Receiving goods: after payment is made, the buyer receives the goods from the vendor, completing the procurement process.

Every company has its own Procure-to-Pay variant: each business handles requests, pricing, and terms in a unique way [99]. Typically, the end-to-end process involves different systems, which are usually not efficiently connected with each other. This results in inconsistent data across the entire chain [101]. For instance, requests may be sent via email and then manually entered into management systems by employees, resulting in potential data entry errors. Price information may be outdated or different from that provided by the supplier. Additional terms and conditions may be included in certain systems but not in others. Purchase requisitions may be approved by users who are not designated for such activities, causing potential issues. These deviations from the desired process can result in significant losses, ranging from financial

penalties to the payment of incorrect prices for wrong quantities of products. Additionally, companies waste a considerable amount of time on manual work to correct inconsistencies. While errors can not be completely avoided, they can be minimized.

Internal auditing plays a crucial role in ensuring consistency of information throughout the chain and compliance with relevant regulations [27]. It identifies areas where errors occur and quantifies financial and time losses resulting from these inefficiencies. By addressing these issues, companies can enhance their Procure-to-Pay processes. This helps organizations reduce financial risks, improve overall efficiency, control costs, and maintain positive supplier relationships. Automation and technology solutions, such as procurement software or Enterprise Resource Planning (ERP) systems, can be used to enhance efficiency and effectiveness of the P2P process. Moreover, process mining techniques can provide smart-insights based on simulation and predictive monitoring that may increase audit efficiency [102].

3.2 SAP Standard Tables

Systems, Applications, and Products in Data Processing (SAP) is one of the most widely used Enterprise Resource Planning (ERP) software by companies. ERP systems are designed to effectively collect, organize, and store business processes, facilitating the continuous and centralized flow of data. Within SAP, data management is accomplished through a standardized set of tables that encompass various fields and levels of detail. A SAP table refers to a database table that stores specific data, such as master data, transactional data, and configuration data, within a SAP system [103]. Each one of them corresponds to a specific entity or object within the SAP system, such as customers, vendors, materials, purchase orders, sales orders, etc. Tables are interconnected with each other through foreign keys to establish relationships between different entities. For instance, some tables store high-level information about orders, while being connected to tables that contain detailed information about each individual order. Usually companies do not use all available SAP tables and they often customise them to meet specific requirements.

For the creation of the dataset used in this work, we take inspiration from different SAP table. We build tables EKKO, EKPO, EBAN, RSEG, RBKP, EKDS, JSEG, P0002, and MARA. It follows a brief description of each table:

- EKKO: header of the *Purchase Orders* (POs). It contains high-level information about orders.

- EKPO: it contains details of the Purchase Orders Item present in EKKO.
- EBAN: it contains details of the Purchase Requisitions.
- RBKP: header of the invoices. It contains high-level information about invoices.
- RSEG: it contains details of the invoices present in RBKP.
- EKDS: custom table that contains the changing in price or quantity of the order in EKPO.
- JSEG: custom table that contains the changing in status of the order items.
- P0002: staff master data. It contains information about employees.
- MARA: it contains information about the material that can be bought.

Some of the table fields used are the standard one provided by SAP, whereas others were invented on purpose.

3.3 Dataset creation

We simulate the dataset of an industrial company divided into six plants, referring to workplaces situated in different countries. The P2P process of the company concerns the purchase of materials for production. We begin the simulation of the dataset by creating tables P0002 and MARA.

We consider 100 different materials, categorized into 8 material groups. Each material is associated with one of the 19 different vendors. Additionally, each material has certain attributes, including a standard price, expected days for shipment, and expected days for delivery. A material can be *free-pass* or not. The former means that it can be purchased without the need of a PR, whereas the latter means the contrary. Using this information we create the MARA table, whose first rows can be seen in Tab. 3.1.

The P0002 table contains the staff master data. Each employee is assigned to one of the six different plants. Each plant has one designated person, referred to as the *supervisor*, who is responsible for purchase requisitions approval. First rows of P0002 table can be seen in Tab. 3.2.

The two tables just described are created once and kept fixed for any simulation. On the contrary, the others depend on parameters that can be adjusted, allowing some flexibility in

MATNR	MATKL	MEINS	NETPR	PEINH	RKDST	PLIFZ	PLICZ	LIFNR	NAME1
10615744	TPM1	PCS	5	EUR	NA	2	8	30003467	Office Supplies
11159536	TPM1	PCS	1	EUR	NA	2	6	30003467	Office Supplies
11788404	TPM1	PCS	6	EUR	NA	2	8	30003467	Office Supplies
13405872	TPM1	PCS	3	EUR	NA	2	5	30003467	Office Supplies
13505536	TPM1	PCS	7	EUR	NA	2	9	30003467	Office Supplies
15353960	TPM1	PCS	2	EUR	NA	2	7	30002348	Material Delivery S.p.A
16177764	TPM1	KG	4	EUR	NA	2	4	30002348	Material Delivery S.p.A
17231232	TPM1	PCS	5	EUR	NA	2	6	30002348	Material Delivery S.p.A
18608880	TPM1	PCS	1	EUR	NA	2	5	30002348	Material Delivery S.p.A
19297648	TPM1	PCS	2	EUR	NA	2	8	30002348	Material Delivery S.p.A
19474160	COLA	KG	986	EUR	x	10	4	48114031	FeV SRL
19816724	COLA	PCS	521	EUR	x	3	5	32952574	FoodFY
20667944	COLA	KG	463	EUR	x	7	6	73419740	Green Company
21595928	COLA	PCS	83	EUR	x	2	5	43088822	PA Inc.
22818912	COLA	KG	636	EUR	x	10	5	50759436	O.R.T.O.

Table 3.1: First rows of the MARA table. Fields are defined as follows: material ID (MATNR), material group (MATKL), base unit of measure (MEINS), net price in purchasing document (NETPR), price unit (PEINH), PR needed indicator (RKDST), expected days for shipment (PLIFZ), expected days for deliver (PLICZ), vendor ID (LIFNR), vendor name (NAME1).

PERNR	ERNAM	NAME2	NACHN	EMAIL	WERKS	MATKL
22179495	Marta Lombardi	Marta	Lombardi	marta.lombardi@demo.it	US10	COLA
53251766	Andrea Bruno	Andrea	Bruno	andrea.bruno@demo.it	BR10	COLA
23246292	Marco Bianchi	Marco	Bianchi	marco.bianchi@demo.it	CN10	COLA
33607135	Sara Romano	Sara	Romano	sara.romano@demo.it	JP11	COLA
63821277	Gabriele Russo	Gabriele	Russo	gabriele.russo@demo.it	DE10	COLA
30090592	Giulia Russo	Giulia	Russo	giulia.russo@demo.it	IN10	COLA
84141246	Luca Gallo	Luca	Gallo	luca.gallo@demo.it	BR10	EDSA
23329589	Sara Russo	Sara	Russo	sara.russo@demo.it	US10	EDSA
49485597	Anna Costa	Anna	Costa	anna.costa@demo.it	CN10	EDSA
58569539	Gabriele Rossi	Gabriele	Rossi	gabriele.rossi@demo.it	DE10	EDSA
59231227	Gabriele Romano	Gabriele	Romano	gabriele.romano@demo.it	JP11	EDSA
86162838	Giulia Moretti	Giulia	Moretti	giulia.moretti@demo.it	IN10	EDSA
49826393	Sara Costa	Sara	Costa	sara.costa@demo.it	IN10	MPLF
20837115	Riccardo Fontana	Riccardo	Fontana	riccardo.fontana@demo.it	CN10	MPLF

Table 3.2: First rows of the P0002 table that shows the supervisors. Fields are defined as follows: employee ID (PERNR), employee name (ERNAM), employee first name (NAME2), employee last name (NACHN), employee email (EMAIL), plant ID (WERKS), material group of the supervisor (MATKL).

the simulation and the possibility of generating datasets with varying characteristics. Some of these variables include the number of orders to be simulated, the range of dates of the simulation, the number of items per order, and the anomalies ratio. The latter determines frequency or likelihood of errors occurring within simulated data, such as incorrect pricing, missing information, or other inconsistencies.

Following we report the simulation process of the dataset:

- Purchase Orders (POs): we simulate 5000 unique orders with dates ranging from January 1, 2019, to December 31, 2022.
- Items per order: for each order, we determine the number of items it contains by randomly sampling values from a beta distribution. This introduce variability in the number of items present in each order.
- Quantity ordered: for each item, we assign a random integer between 1 and 100 representing the quantity ordered.
- Item price: for each item, we retrieve the item price from the MARA table.
- Employee and plant: we associate each order with the employee who created it and with the employee's plant.
- Purchase Requisitions (PRs): for each item that needs it, we generate the purchase requisition. These are created by gathering items based on material group, plant, and creation date. Each requisition is associated with the name of the supervisor responsible for approval.
- PR creation date: for each item that needs it, we set the PR creation date as the day after the order creation date.
- Change status date: for each item that needs the PR, we determine the change status date (rejected or approved) as a random number of days after the PR creation date.
- Sent to vendor date: we determine the date when the PO is sent to the vendor based on a random number of days after the change status date. If the material does not require a PR (free-pass material), this date coincides with the order creation date.
- Shipment and delivery: we simulate the shipping and delivery of items. For each item, we choose dates as random days around the expected shipping and delivery days, both of which are contained in the MARA table.

By following these steps, we populate the EKKO, EKPO, and EBAN tables. First rows of EKPO and EBAN tables can be seen in Tab. 3.3 and Tab. 3.4.

Regarding the invoicing process, we follow these steps:

EBELN	EBELP	MATNR	MENGE	DELGE	MEINS	NETPR	PEINH	LCWED	SEDAT
1000015	1	13505536	87	87	PCS	7	EUR	2019-01-21	2019-01-08
1000010	1	16177764	84	83	KG	4	EUR	2019-01-25	2019-01-21
1000008	1	11788404	12	12	PCS	6	EUR	2019-02-05	2019-01-24
1000008	2	13405872	18	18	PCS	3	EUR	2019-02-05	2019-01-24
1000008	3	16177764	79	79	KG	4	EUR	2019-02-05	2019-01-28
1000007	1	17231232	29	29	PCS	5	EUR	2019-01-29	2019-01-21
1000003	1	10615744	52	52	PCS	5	EUR	2019-02-05	2019-01-24
1000012	1	18608880	39	39	PCS	1	EUR	2019-02-12	2019-01-31
1000012	2	10615744	97	97	PCS	5	EUR	2019-02-12	2019-01-31
1000017	1	19297648	56	56	PCS	2	EUR	2019-02-12	2019-01-31
1000005	1	75603272	80	80	PCS	320	EUR	2019-02-05	2019-01-24
1000006	1	11788404	71	71	PCS	6	EUR	2019-02-08	2019-01-24
1000014	1	16177764	98	98	KG	4	EUR	2019-02-01	2019-01-28
1000016	1	17231232	64	64	PCS	5	EUR	2019-02-06	2019-01-28
1000009	1	13405872	76	76	PCS	3	EUR	2019-01-22	2019-01-15

Table 3.3: First rows of a partial EKPO table. Fields are defined as follows: PO number (EBELN), PO item number (EBELP), material number (MATNR), ordered quantity (MENGE), delivered quantity (DELGE), PO UOM (MEINS), material net price (NETPR), price unit (PEINH), latest possible goods receipt (LCWED), shipping date (SEDAT).

BANFN	BNFPO	BADAT	EBELN	EBELP	BSAKZ	MENGE	MEINS	NETPR	PEINH
467282	1	2019-01-02	1000001	3	x	79	KG	618	EUR
168159	1	2019-01-02	1000002	3	x	93	PCS	915	EUR
567413	1	2019-01-02	1000002	4	x	46	PCS	605	EUR
188381	1	2019-01-02	1000003	2	x	49	PCS	916	EUR
370400	1	2019-01-02	1000004	2	x	86	PCS	159	EUR
535829	1	2019-01-03	1000004	2	x	90	PCS	159	EUR
478603	1	2019-01-02	1000005	1	x	79	PCS	320	EUR
217952	1	2019-01-03	1000005	1	x	80	PCS	320	EUR
885684	1	2019-01-02	1000005	2	x	62	PCS	605	EUR
148540	1	2019-01-03	1000006	2	x	89	PCS	845	EUR
252315	1	2019-01-14	1000006	2	x	89	PCS	853	EUR
204181	1	2019-01-03	1000006	3	x	86	KG	702	EUR
714809	2	2019-01-04	1000009	2	x	73	PCS	159	EUR
919363	1	2019-01-04	1000010	2	x	64	PCS	554	EUR
459783	1	2019-01-08	1000010	2	x	63	PCS	554	EUR

Table 3.4: First rows of the EBAN table. Fields are defined as follows: PR number (BANFN), PR item number (BNFPO), PR date (BADAT), PO number (EBELN), PO item number (EBELP), PR control indicator (BSAKZ), ordered quantity (MENGE), PO UOM (MEINS), material net price (NETPR), price unit (PEINH).

- Invoice creation: we create invoices grouping POs by weeks, vendor, and plant. Invoices refer to POs from the previous week.
- Invoice receipt date: we randomly assign the invoice receipt date as Monday, Tuesday, or Wednesday. This introduces variability in the invoice reception.
- Invoice expiration date: we set the expiration date of each invoice as 21 days after the invoice receipt date.
- Invoice payment date: we randomly determine the invoice payment date, which occurs around the expiration date.

For each invoice, we extract details such as quantity, price, and plant from the related order in the EKPO table. This ensures consistency between invoices and the corresponding orders. The steps previously described populate the RSEG and the RBKP tables. First rows of RSEG table can be seen in Tab. 3.5.

BELNR	BUZEI	EBELN	EBELP	MATNR	MENGE	BSTME	NETPR	PEINH	WERKS
5329562	1	1000015	1	13505536	87	PCS	7	EUR	CN10
9552201	3	1000004	1	18608880	67	PCS	1	EUR	CN10
5665158	1	1000011	1	17231232	23	PCS	5	EUR	DE10
7970906	2	1000002	2	11159536	69	PCS	1	EUR	IN10
6250996	3	1000002	1	18608880	12	PCS	1	EUR	IN10
9332695	1	1000001	1	11788404	69	PCS	6	EUR	JP11
8799348	1	1000009	1	13405872	76	PCS	3	EUR	DE10
7421958	3	1000016	1	17231232	64	PCS	5	EUR	BR10
7421958	2	1000014	1	16177764	98	KG	4	EUR	BR10
5019336	4	1000006	1	11788404	71	PCS	6	EUR	BR10
9332695	2	1000001	2	13405872	4	PCS	3	EUR	JP11
6250996	2	1000017	1	19297648	53	PCS	2	EUR	IN10
7970906	1	1000012	2	10615744	97	PCS	5	EUR	IN10
6250996	1	1000012	1	18608880	39	PCS	1	EUR	IN10
5019336	3	1000003	1	10615744	52	PCS	5	EUR	BR10

Table 3.5: First rows of the RSEG table. Fields are defined as follows: invoice number (BELNR), invoice item number (BUZEI), PO number (EBELN), PO item number (EBELP), material number (MATNR), ordered quantity (MENGE), PO UOM (BSTME), material net price (NETPR), price unit (PEINH), plant ID (WERKS).

3.3.1 Anomalies simulation

Until now, we have simulated data representing the ideal process. However, to reflect real-world scenarios, we need to introduce anomalies that affect the data. These include:

- Three way mismatch: we simulate changes in quantities and prices that are not always updated in every table. This can result in inconsistencies between invoices and orders. The EKDS table captures these changes.
- Missing Purchase Requisitions: we delete a certain number of entries from the EBAN table. This indicates that some PRs are missing.
- Purchase Requisitions rejected but item bought: we define specific PR to be rejected, even though the corresponding articles are still bought.
- Late payments, shipments, and deliveries: invoices and goods may be paid, shipped, and delivered late, deviating from the expected timelines.

Second and third anomalies refer to the name of *Maverick Buying*. A Maverick Buying occurs when an organization makes a purchase without adhering to predetermined terms and procedures [104]. It typically involves bypassing authorized purchasing channels and making direct purchases from unauthorized suppliers or vendors.

The amount of these anomalies vary according to plant. Tab. 3.6 reports the anomaly ratios used for the simulation of our dataset. Rates define different types of events and anomalies. First of all, they determine the number of orders that go through "Change Price" and "Change Quantity" activities. Secondly, they characterize the possible discordance between values contained in tables, the ratio of deleted and rejected purchase requisitions, the ratio of Maverick Buying, and the delay of the invoice payment.

PLANT	ANOMALY RATE
IN10	30%
US10	25%
DE10	20%
CH10	15%
BR10	10%
JP11	1%

Table 3.6: Anomaly rate for each plant.

3.3.2 Activity Tables

The main ingredient of process mining is the event log, which we refer to as *Activity Table* in the Celonis platform. We create Activity tables by associating each activity with its execution time and by ordering them. Activities are not explicitly present in enterprises datasets, but they

need to be defined by a process expert, as already discussed in Sec. 1.1.2. Tab. 3.7 contains the list of possible activities of our model and the fields involved in their definition.

An Activity table needs to be related to a Case table, which contains all unique Case IDs. In our dataset, the Case tables are EKPO and RSEG. Therefore, we create two Activity tables:

- Activity table for Purchase Orders (_CEL_PP_ACTIVITIES): this table captures events related to purchase orders, such as the PR approval cycle, and shipment and delivery from the vendor. The table includes information such as activity ID, order ID, activity description, its timestamp, employee involved, plant, and any other relevant detail. First rows of the table are shown in Tab. 3.8.

- Activity table for Invoices (_CEL_PP_ACTIVITIES_INVOICES): this table captures events related to the invoices process. It may include information such as activity ID, invoice ID, activity description, its timestamp, etc. First rows of the table are shown in Tab. 3.9.

These Activity tables help track and analyze actions and events that occur within the purchase order and invoicing processes. They provide a comprehensive view of the activities performed, allowing for better process understanding, analysis of bottlenecks or inefficiencies, and identification of patterns or anomalies.

After an initial simulation of the first set of data, we implement an online orders creation by setting up a script that runs once a day. By doing so, the simulation can accurately represent the dynamic nature of a real-time P2P process. While we simulate future dates in the back-end environment, we do not report them in the front-end to maintain a realistic online process. This allows the creation of *open traces*, which represent orders that are still in progress and have not been completed yet. As each day passes, we reevaluate and carry dates in the front-end environment accordingly. This ongoing simulation helps capture the evolving nature of the process, enabling analysis of real-time data and evaluation of performance metrics on a day-to-day basis.

3.3.3 Data Model

The process of creating the event log is not the final step in the data load procedure in a process mining platform. We need to indicate relationships between tables by connecting their fields through primary and foreign keys. This enables the use of queries that define, for example, Key Performance Indicators (KPIs), Key Risk Indicators (KRIs), or Filters that can propagate

ACTIVITY	TABLE	FIELDS	TIMESTAMP FIELD	DESCRIPTION
Create Purchase Order Item	EKKO EKPO	EKKO.WERKS EKKO.EBELN EKPO.EBELP	EKKO.AEDAT	Order creation
Create Purchase Requisition Item	EKKO EKPO EBAN	EKKO.WERKS EKKO.EBELN EKPO.EBELP	EBAN.BADAT	PR creation
Purchase Requisition Item: Accepted	EKKO EKPO EBAN	EKKO.WERKS EKKO.EBELN EKPO.EBELP EBAN.BSAKZ	EBAN.ERDAT	PR accepted when EBAN.BSAKZ = x
Purchase Requisition Item: Rejected	EKKO EKPO EBAN	EKKO.WERKS EKKO.EBELN EKPO.EBELP EBAN.BSAKZ	EBAN.ERDAT	PR rejected when EBAN.BSAKZ $\neq$ x
Change Price	EKKO EKPO EKDS	EKKO.WERKS EKKO.EBELN EKPO.EBELP EKDS.NETPR	EKDS.UDATE	Change price of the item when EKDS.NETPR not null
Change Quantity	EKKO EKPO EKDS	EKKO.WERKS EKKO.EBELN EKPO.EBELP EKDS.MENGE	EKDS.UDATE	Change quantity of the item when EKDS.MENGE not null
Purchase Order Item: Sent to Vendor	EKKO EKPO	EKKO.WERKS EKKO.EBELN EKPO.EBELP	EKPO.DRDAT	PR accepted (if necessary) and item sent to vendor
Late Shipment	EKKO EKPO	EKKO.WERKS EKKO.EBELN EKPO.EBELP EKPO.SEDAT	EKPO.LEWED	Item late when the expected shipment date EKPO.LEWED is earlier than the shipment date EKPO.SEDAT
Goods Shipped	EKKO EKPO	EKKO.WERKS EKKO.EBELN EKPO.EBELP	EKPO.SEDAT	Item shipped
Late Delivery	EKKO EKPO	EKKO.WERKS EKKO.EBELN EKPO.EBELP EKPO.ITDAT	EKPO.LCWED	Item late when the expected delivery date EKPO.LCWED is earlier than the delivery date EKPO.ITDAT
Record Goods Issue	EKKO EKPO	EKKO.WERKS EKKO.EBELN EKPO.EBELP	EKPO.ITDAT	Item received
Record Invoice	RBKP RSEG	RBKP.WERKS RBKP.BELNR RSEG.BUZEI	RBKP.BLDAT	Invoice received
Due Date Passed	RBKP RSEG	RBKP.WERKS RBKP.BELNR RSEG.BUZEI RBKP.BPDAT	RBKP.BUDAT	Overdue invoice when the expiration date RBKP.BUDAT is earlier than the payment date RBKP.BPDAT
Cleared Invoice	RBKP RSEG	RBKP.WERKS RBKP.BELNR RSEG.BUZEI	RBKP.BPDAT	Invoice paid

Table 3.7: Activity creation. The first column refers to the activity name, the second and the third to the tables and their fields involved in the activity, respectively. The fourth column refers to the field taken into account for the timestamp definition and the last column provides a description of the activity and how it is defined.

CASE KEY	WERKS	EBELN	EBELP	ACTIVITY	EVENTTIME	SORTING
BR10-1000003-1	BR10	1000003	1	Create Purchase Order Item	2019-01-01	10000
BR10-1000003-1	BR10	1000003	1	Change Quantity	2019-01-01	11000
BR10-1000003-1	BR10	1000003	1	Purchase Order Item: Sent to Vendor	2019-01-01	15000
BR10-1000003-1	BR10	1000003	1	Late Shipment	2019-01-18	21000
BR10-1000003-1	BR10	1000003	1	Goods Shipped	2019-01-24	20000
BR10-1000003-1	BR10	1000003	1	Record Goods Issue	2019-02-05	23000
BR10-1000003-2	BR10	1000003	2	Create Purchase Order Item	2019-01-01	10000
BR10-1000003-2	BR10	1000003	2	Create Purchase Requisition Item	2019-01-02	12000
BR10-1000003-2	BR10	1000003	2	Purchase Requisition Item: Approved	2019-01-03	13000
BR10-1000003-2	BR10	1000003	2	Purchase Order Item: Sent to Vendor	2019-01-08	15000
BR10-1000003-2	BR10	1000003	2	Goods Shipped	2019-01-18	20000
BR10-1000003-2	BR10	1000003	2	Record Goods Issue	2019-02-05	23000
BR10-1000006-1	BR10	1000006	1	Create Purchase Order Item	2019-01-02	10000
BR10-1000006-1	BR10	1000006	1	Change Quantity	2019-01-02	11000
BR10-1000006-1	BR10	1000006	1	Purchase Order Item: Sent to Vendor	2019-01-02	15000

Table 3.8: First rows of _CEL_PP_ACTIVITIES table. Fields are defined as follows: case ID (CASE KEY), plant ID (WERKS), PO number (EBELN), PO item number (EBELP), activity name (ACTIVITY), activity time (EVENTTIME), activity sorting order (SORTING).

CASE KEY	WERKS	BELNR	BUZEI	ACTIVITY	EVENTTIME	SORTING
BR10-4936455-1	BR10	4936455	1	Record Invoice	2019-12-03	10200
BR10-4936455-1	BR10	4936455	1	Cleared Invoice	2019-12-17	10400
BR10-4941636-1	BR10	4941636	1	Record Invoice	2019-12-30	10200
BR10-4941636-1	BR10	4941636	1	Cleared Invoice	2020-01-10	10400
BR10-4943181-1	BR10	4943181	1	Record Invoice	2020-10-27	10200
BR10-4943181-1	BR10	4943181	1	Cleared Invoice	2020-11-12	10400
BR10-4943181-2	BR10	4943181	2	Record Invoice	2020-10-27	10200
BR10-4943181-2	BR10	4943181	2	Cleared Invoice	2020-11-12	10400
BR10-4943181-3	BR10	4943181	3	Record Invoice	2020-10-27	10200
BR10-4943181-3	BR10	4943181	3	Cleared Invoice	2020-11-12	10400
BR10-4943181-4	BR10	4943181	4	Record Invoice	2020-10-27	10200
BR10-4943181-4	BR10	4943181	4	Cleared Invoice	2020-11-12	10400
BR10-4943181-5	BR10	4943181	5	Record Invoice	2020-10-27	10200
BR10-4943181-5	BR10	4943181	5	Cleared Invoice	2020-11-12	10400
BR10-4943181-6	BR10	4943181	6	Record Invoice	2020-10-27	10200

Table 3.9: First rows of _CEL_PP_ACTIVITIES_INVOICES table. Fields are defined as follows: case ID (CASE KEY), plant ID (WERKS), invoice number (BELNR), invoice item number (BUZEI), activity name (ACTIVITY), activity time (EVENTTIME), activity sorting order (SORTING).

across every table that contains affected objects. The set of correlations between fields is referred to as the *data model*. Loading the data model ensures consistency across data, including uniqueness of primary keys and establishment of one-to-many ($1:N$) relationships. The data model resulting from the simulated dataset can be seen in Fig. 3.1.

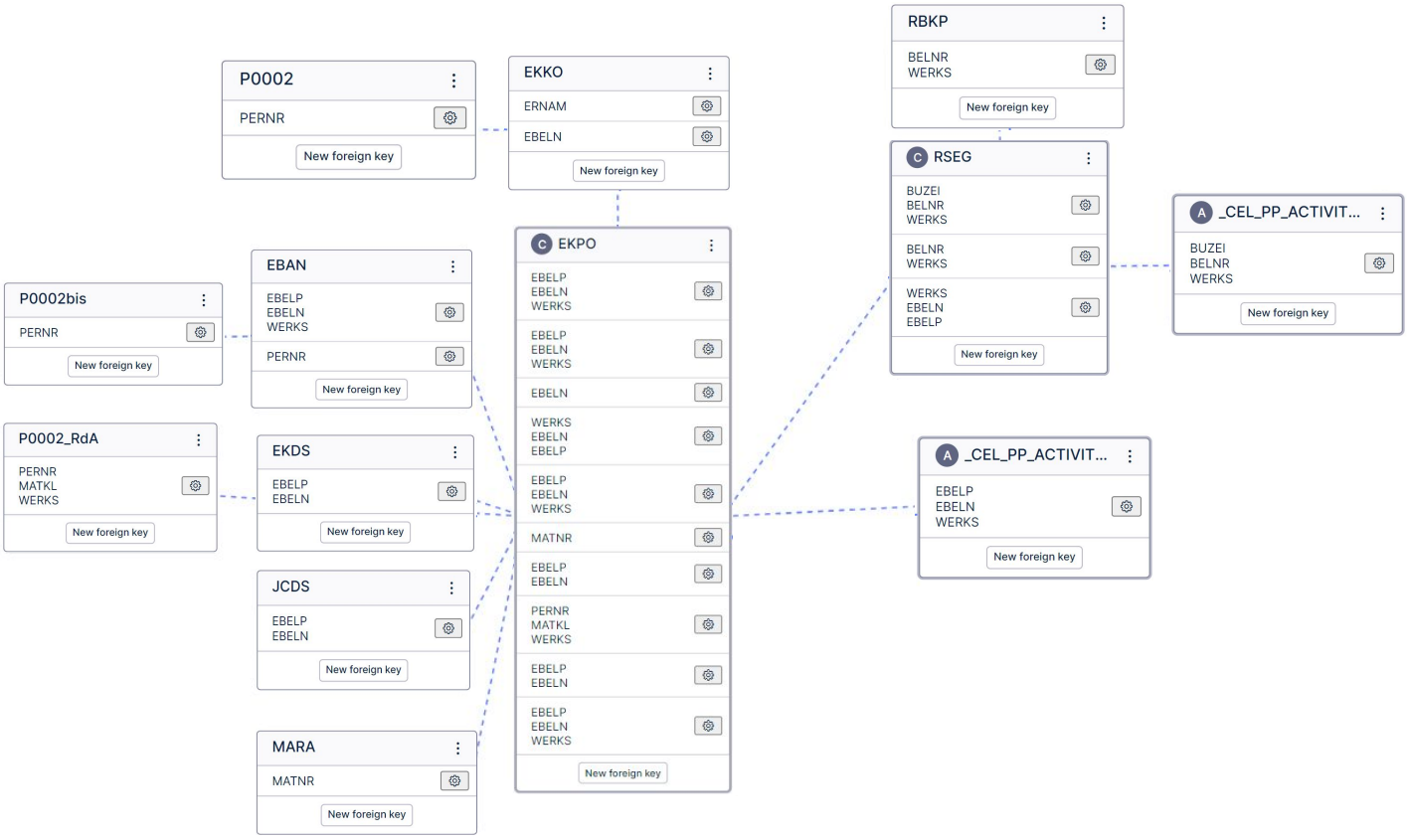

Figure 3.1: Data model for the simulated dataset. In the center there is the Case table EKPO, which is connected with almost every other tables with $1:N$ and $N:1$ connections. The second Case table is the RSEG, which is connected with the invoices Activity table, RBKP and EKPO. Table P0002 is present three times with different names in order to allow the creation of some particular queries.

3.3.4 Process Explorer and Variant Explorer

Using the Celonis tool called Process Explorer, we can obtain a clear and comprehensible picture of the resulting processes. One of its main characteristics is flexibility to expand or collapse the number of activities and links displayed. Additionally, it allows us to connect multiple processes together and visualize them in a user-friendly picture. An example of a process view of

the dataset given by the Process Explorer is shown in Fig. 3.2.

Another useful Celonis feature is the Variant Explorer, which enables to select and visualize specific process variants. The first variant represents the most frequently replayed trace and usually the ideal process. In our case, as shown in Fig. 3.3, the first variant reflects the ideal process flow of not free-pass materials.

By exploiting the visual capabilities of the Celonis tools, we get a simplified view of the ideal process:

1. An employee creates an order, which can contain multiple items. Some items may require a purchase requisition (PR) before they can be bought.
2. The ordered quantity or material price may change during the process. Quantity changes can occur due to internal requirements, while price changes can result from seller updates of the material price list. If a material is not free-pass, any quantity or price change requires a new PR, deprecating the previous one. Items can only be purchased if the last valid PR is approved, otherwise the purchase cannot proceed.
3. Once the supervisor approves the item, the order is sent to the vendor for processing.
4. Vendors share with the buyer invoices related to items grouped by week. The company has 21 days to pay these invoices.
5. Vendors typically wait for all items ordered during the previous week before shipping the goods.

Different variants can exist within the process. For example, an item may be sent to vendors after a change in price or quantity, or even after the PR is rejected. These variations reflect the potential complexities and deviations that can occur within the procurement process in reality.

3.4 Features encoding

Features present in a business dataset can be categorized as either categorical or numerical.

On the other hand, we can encode categorical features using one-hot encoding or embedding techniques. In our experiments, we test both encoding methods. One-hot encoding purpose is to represent each category as a binary vector, where all elements are zero except for the index corresponding to the category, which is set to one. Using this technique, a sequence of activities such as $\langle$ A, B, C $\rangle$ is encoded as:

$$\overbrace{[1,0,0]}^{\text{A}}, \overbrace{[0,1,0]}^{\text{B}}, \overbrace{[0,0,1]}^{\text{C}}].$$

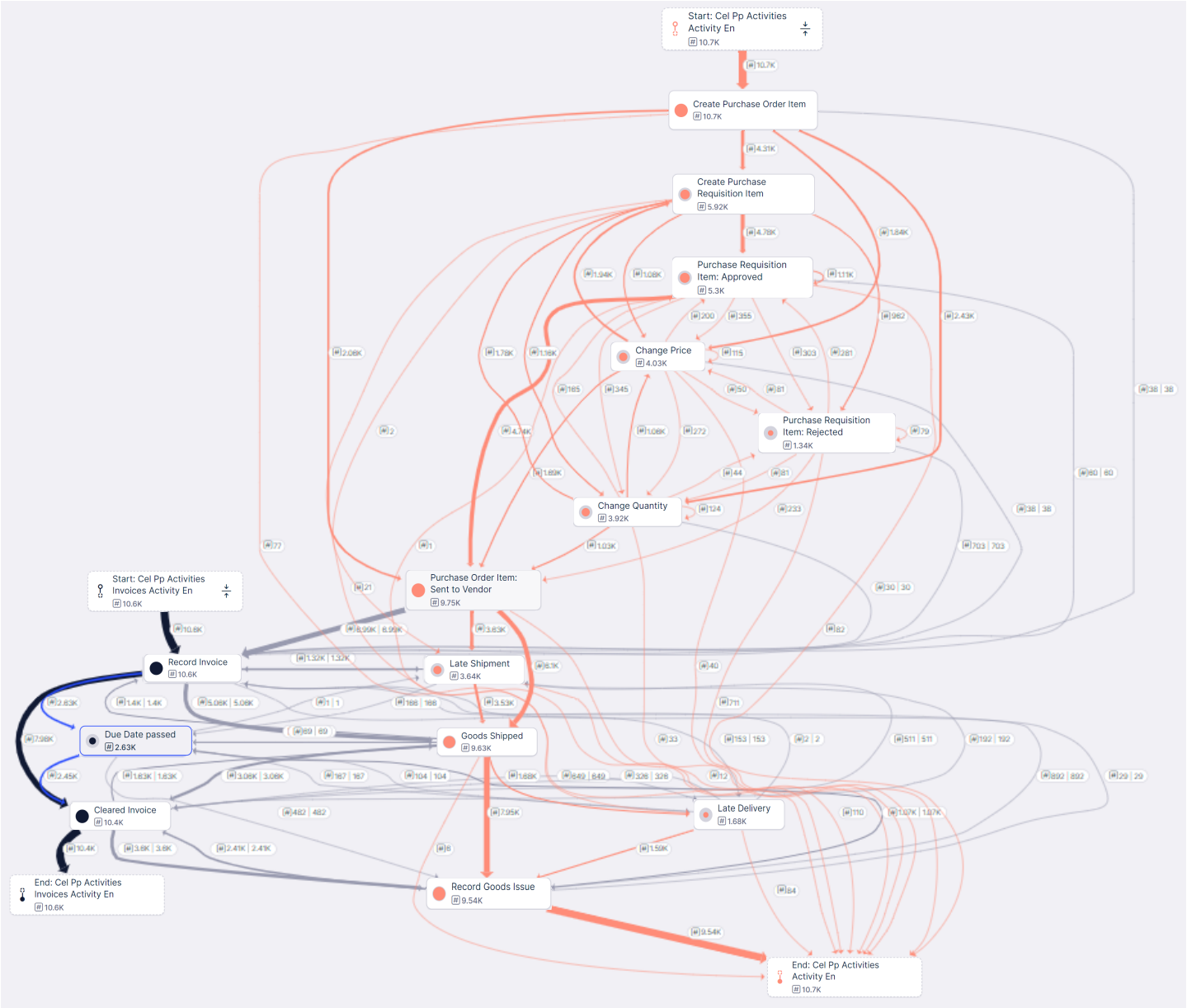

Figure 3.2: Process Explorer example of the simulated dataset. Different colors refer to different processes. It is possible to choose to "close" one process in order to visualize only the other one. This capability is particularly valuable for investigating different departments within a company and understanding their specific subprocesses. In this particular view we show all the activities and links of the simulated dataset.

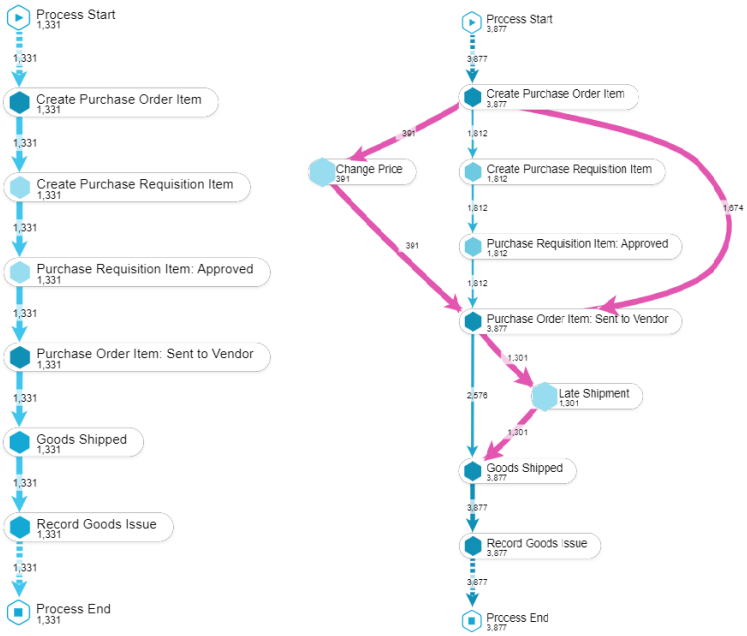

Figure 3.3: Variant Explorer example of the simulated dataset. The left panel shows the first variant, whereas the right one shows variants from 1 to 5. It is possible to select which variants one wants to display.

However, this data representation only captures categorical information and does not consider temporal aspect, such as the time duration between activities.

Instead, we use embedding to convert categorical features into a fixed-length vector of defined size. In many cases, it is impractical to use one-hot encoding for features with a large number of unique values, as it leads to sparse matrices. For example, in a NLP problem the vocabulary size is typically very large. In this case, embedding techniques enable to capture relevant information while reducing data dimensionality, allowing for more efficient and effective processing and analysis [105]. In Tab. 3.10 we report the cardinalities of the features used in our work.

FEATURE	CARDINALITY
Activity	11
Purchase Requisition (PR)	2
Plant ID	6
Vendor ID	19
Material Group	8
Material ID	100

Table 3.10: Categorical features used and their cardinality.

Regarding the embedding encoding for categorical features, we test two different approaches. In the first one, we adopt a separate neural network for each category. Neural networks take the integer encoded categories as input and pass it through a Keras [106] `Embedding` layer with an output dimension of $(1, out\_dim)$. Then, networks reshape the output to $(, out\_dim)$ and concatenate it with all the others. Finally, the resulting vector is fed into the main neural network for the prediction task. In the second approach, we concatenate all categorical features and then we pass the resulting vector to an `Embedding` layer with an output dimension of $(n\_categories, out\_dim)$, then reshaped to $(, n\_categories \times out\_dim)$.

Embedding layers are set to *trainable* = *True*, which means that they are updated during the training process. The layers capture relationships, dependencies, and similarities between different categorical values based on the output of the entire neural network, without relying on one-hot encoding. For example, we consider a dataset that contains the annual income for individuals. We have additional categorical features such as gender and region. These two categories may not have a direct relationship with each other, but they are correlated once considering the third variable, the total income. Natural Language Processing problems usually rely on embedding methods, since categorical variables such as words or tokens, need to be embedded into lower-dimensional representations because of the vocabulary size.

3.5 Resulting dataset

For the purpose of the experiments conducted in this work, we select a series of field from the created tables:

- Case ID: a join of Plant ID, PO Number and PO Item Number.
- Activity Name: name that characterizes the activity.
- Eventtime: time at which the activity is performed.
- Sorting: order of the activity.
- User Name: name of the employee responsible of the activity.
- RKDST: flag for free-pass material.
- NAME1: Vendor ID.
- ELIKZ: flag for closed or open trace.

- MATKL: item material group.
- MATNR: item material number.
- MAVP: flag for Maverick Buying anomaly.
- TWMS: flag for three way mismatch anomaly.

We merge the two processes together since we want to consider them as subprocesses of the same one. Some activities may not refer to a specific action or event, but they are informative. In this way, an end user can have a rapid overview of event information. For example, "Late Shipment", "Late Delivery" and "Due Date passed" activities occur if the dataset records a particular combination of fields. Nevertheless, in any case, no data nor field is recorded following the occurrence of these events. Therefore, we map these activities in features for prediction purposes and remove them from the Activity table. Then, we separate the closed orders from the open ones. The dataset resulting from our simulation is composed as follows: 10409 PO items (process instances) and 93028 activities for the closed orders, 198 PO items and 1363 activities for the open orders.

Then, we proceed to the encoding phase. We map each trace into a one-hot encoded matrix and each case attribute into either a one-hot encoding or a categorical encoding, depending on the performed experiment. We transform the activity timestamp information in a difference in time between the previous activity and the actual one, and next we scale it using `StandardScaler`.

Neural networks for predictive process monitoring aim at learning prediction for open traces, therefore we transform traces into *partial traces*. In this way, the network is capable of learning dependencies in every step of the process. For each trace of length n, we create a partial trace for each windows $seq[0] : seq[k]$ with $0 \leq k \leq n-1$ and $seq[k+1]$ as target. For example, if we have a trace $\langle$ A,B,C,D $\rangle$ with $n = 4$, the partial traces resulting are $\langle$ A $\rangle$ with target B, $\langle$ A,B $\rangle$ with target C, and $\langle$ A,B,C $\rangle$ with target D. In terms of network inputs, we transform the full trace:

$$[\overbrace{[1,0,0,0]}^{\text{A}}, \overbrace{[0,1,0,0]}^{\text{B}}, \overbrace{[0,0,1,0]}^{\text{C}}, \overbrace{[0,0,0,1]}^{\text{D}}]$$

into:

$$\overbrace{[[1,0,0,0]]}^{\text{A}} \longrightarrow \overbrace{[0,1,0,0]}^{\text{B}}$$

$$\overbrace{[[1,0,0,0]}^{\text{A}}, \overbrace{[0,1,0,0]]}^{\text{B}} \longrightarrow \overbrace{[0,0,1,0]}^{\text{C}}$$

$$\overbrace{[[1,0,0,0]}^{\text{A}}, \overbrace{[0,1,0,0]}^{\text{B}}, \overbrace{[0,0,1,0]]}^{\text{C}} \longrightarrow \overbrace{[0,0,0,1]}^{\text{D}}.$$

By considering every possible partial trace, we create 75050 process instances.

Since each trace has a different length, we need to pad each sample to make it suitable to be used as input by the neural network. We pad each partial trace with respect to the longest one, using vectors of zeros of sizes $(length_{max} - length_{trace}, 11)$, where 11 is the number of different activities and 22 is $length_{max}$. At this point, we separate our dataset in training set of length 45030, validation set of length 15010, and test set of length 15010.

An example of a trace encoding process can be seen in Fig. 3.4

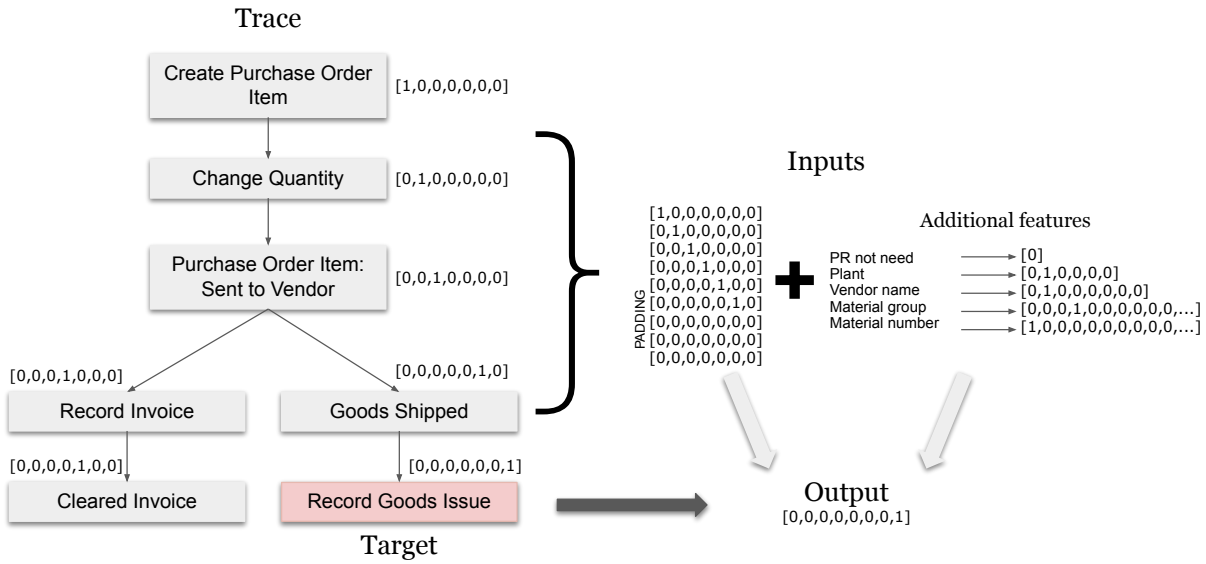

Figure 3.4: Trace encoding using additional attributes. On the left panel is shown the trace and the one-hot encoding of the activities. On the right panel are shown inputs for the neural network, which are the trace encoded and padded and additional features regarding the trace.

4

Experiments and Results

In the second chapter we presented the predictive process monitoring problem. Now we want to apply the deep learning techniques introduced before to the simulated dataset. Experiments are divided into several groups. First of all, we investigate the task of predicting the next activity using different types of neural networks and different architectural configurations. After performing the first set of experiments, we select a specific architecture and we start adding features as inputs to the model and observe how performances vary. We apply different types of encoding methods to additional features. Finally, we explore different prediction tasks, such as predicting the execution time of the next activity and estimating the probability of anomalies.

We perform the experiments using deep learning API Keras. Its Functional API allows us to build arbitrary model architectures with multiple inputs and outputs. We train models for 60 epochs with a batch size of 64, except where indicated otherwise. We select the number of epochs based on the observation that most models reach saturation after few epochs, typically around 20-30. For each experiment, we save the model with the smallest loss function value, evaluated over the validation set across the epochs. Then, we compare results obtained by performing prediction on the test set.

4.1 Next Activity prediction

End users can exploit deep learning models to predict the likelihood of upcoming events for open traces. This allows them to prioritise orders that have a high probability of containing

anomalous activity at the next step. To accomplish this, we focus our task on evaluating prediction probabilities using *scoring rules*, previously illustrated in Sec. 1.2.1. We build a custom weighted version of the *categorical cross-entropy* to be used as loss function and to effectively evaluate models predictions. Weights used in this metric are based on the target activity frequencies in the training set. The approach is particularly beneficial for forecasting uncommon activities since it assigns greater importance to correctly predicting less frequent events. In fact, in process mining these are often associated with anomalies, therefore assigning them greater weight helps improve model performances in detecting such outliers.

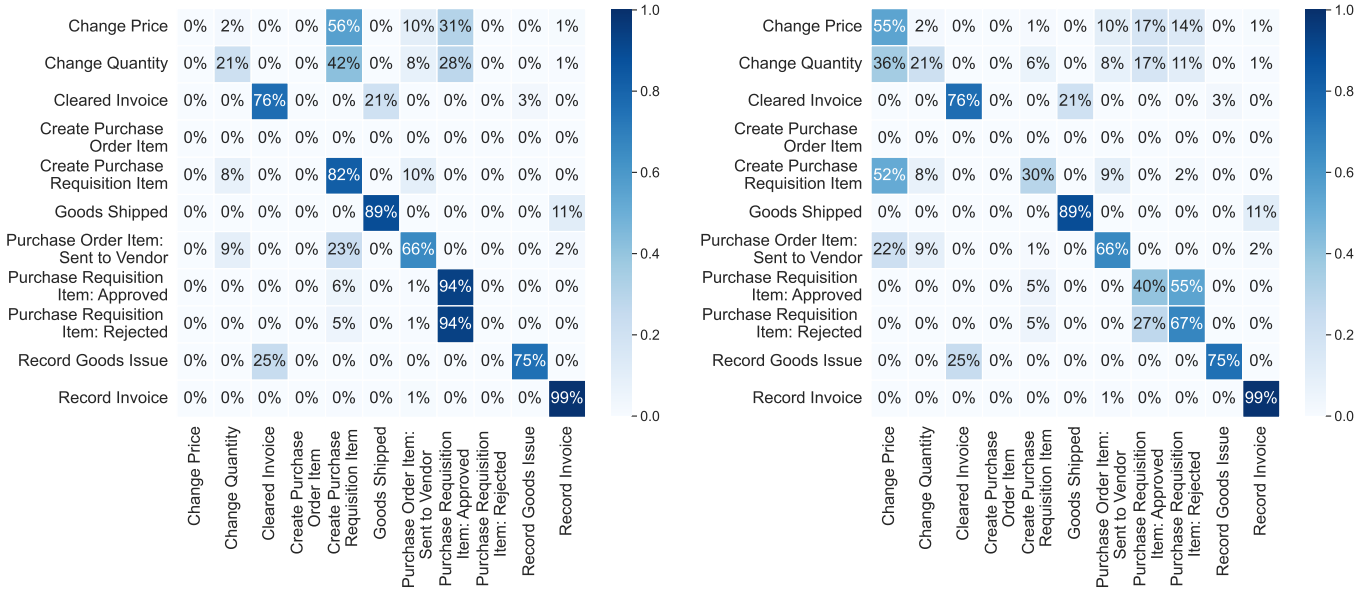

Figure 4.1: Confusion matrices comparison with and without the *weighted* categorical cross-entropy. Matrices are normalized over true labels. The left panel shows the confusion matrix trained using the standard categorical cross-entropy function, whereas on the right panel we can see the prediction of the model trained using the weighted categorical cross-entropy function. The first model never predicts activities "Change Price" and "Purchase Requisition Item: Rejected", which are instead present in the second confusion matrix.

Fig. 4.1 shows a comparison between confusion matrices generated by two models using the same architecture, but trained using different loss functions. The left panel shows the prediction results of the model trained using the standard categorical cross-entropy, while the right panel shows the confusion matrix generated by the model trained using the weighted loss version. The first model hardly predicts non frequent activities such as "Change Price", "Change Quantity" and "Purchase Requisition Item: Rejected". The second model predictions, on the other hand, are more balanced.

In a multiclass classification task, researches mostly use the softmax activation function to

convert neural networks output logits into probability values. However, sometimes this leads to overconfident or miscalibrated predictions. To overcome this issue, we calibrate the neural network using a post-processing technique known as *Temperature Scaling*. This method aims to refine the softmax activation function and make output probabilities more accurately reflect the true likelihood [107]. Given N possible classes, Temperature Scaling transforms the logits o_i in the following probability distribution:

$$P(\hat{\mathbf{y}}) = \frac{e^{\mathbf{o}/T}}{\sum_{i=0}^{N} e^{o_i/T}}, \tag{4.1}$$

where the parameter T is the one that minimizes the loss function.

In this set of experiments, we train networks in the task of predicting the next activity. We explore various aspects, such as changing model architectures, varying the number of neurons per layer, selecting different input features and feature encoding methods. By experimenting different neural network configurations, we aim to compare the most used deep learning approaches. Furthermore, we want to apply the resulting models to open traces in order to provide an online operational supporting framework for end users in a process mining platform.

Selected models have a similar structure. Firstly, the activity input, encoded in one-hot vector, passes through a `Masking` layer, which has the task of masking the padding vectors, and then goes through hidden layers. Finally, the output part consists in a `Dense` layer with 11 nodes, representing the possible activities in the dataset. Then, we apply the softmax activation function, which computes the probability of each activity occurring at the next step in the process. We train models using the weighted categorical cross-entropy loss function. For optimization, we use the Adam optimizer with a learning rate of 0.0001. After the training phase, we evaluate models by applying the Temperature Scaling method over the `Dense` layer output, before the softmax activation function.

4.1.1 Architecture exploration

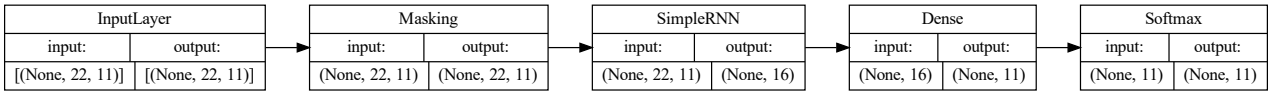

Figure 4.2: RNN_arch_1 model architecture. Other standard model architectures are the same but the `SimpleRNN` layer is replaced by `LSTM` and `GRU` layers. In this particular case, the hidden layer has 16 nodes and the argument `return_sequence` is set to False, since its the last temporal layer.

In the architecture exploration experiment, we investigate the performance of different neural networks for the prediction task. We use architecture such as RNN, LSTM, GRU, and autoencoder-based. An example of a model representing the first three networks categories is illustrated in Fig. 4.2.

Regarding the autoencoder-based network, we first train an autoencoder in the task of reconstructing input sequences. Its first hidden layer is a `SimpleRNN` layer with the parameter `return_sequence` set to True. This makes the layer output one hidden state for each input time step. The bottleneck output is a n-dimensional vector that is repeated using `RepeatVector` layer to match the input dimension. The last hidden layer is the same as the first one. Later there is a `TimeDistributed` layer acting on a `Dense` layer with 11 neurons. In this way, the network produces an output for every single time step, allowing the output to match the input. The autoencoder learns to encode the input into a lower-dimensional representation and then to decode it back to the original format. After the autoencoder training, we extract its bottleneck layer, which captures the compressed representation of the input sequences. Finally, we use this layer as input for a feedforward neural network. The overall process is depicted in Fig. 4.3.

Results

In Tab. 4.1 we provide a summary of the results achieved from the architecture exploration. We report only models that did not show signs of overfitting. These are identified by analyzing the learning curves, one of which can be seen in Fig. 4.4. Neural networks have different number of trainable parameters, but they reach similar performances.

The second autoencoder-based architecture achieves the lowest loss value among the other models, but the total number of trainable parameters is higher. In addition, autoencoder-based models require to be trained for more epochs before reaching loss function saturation. On the

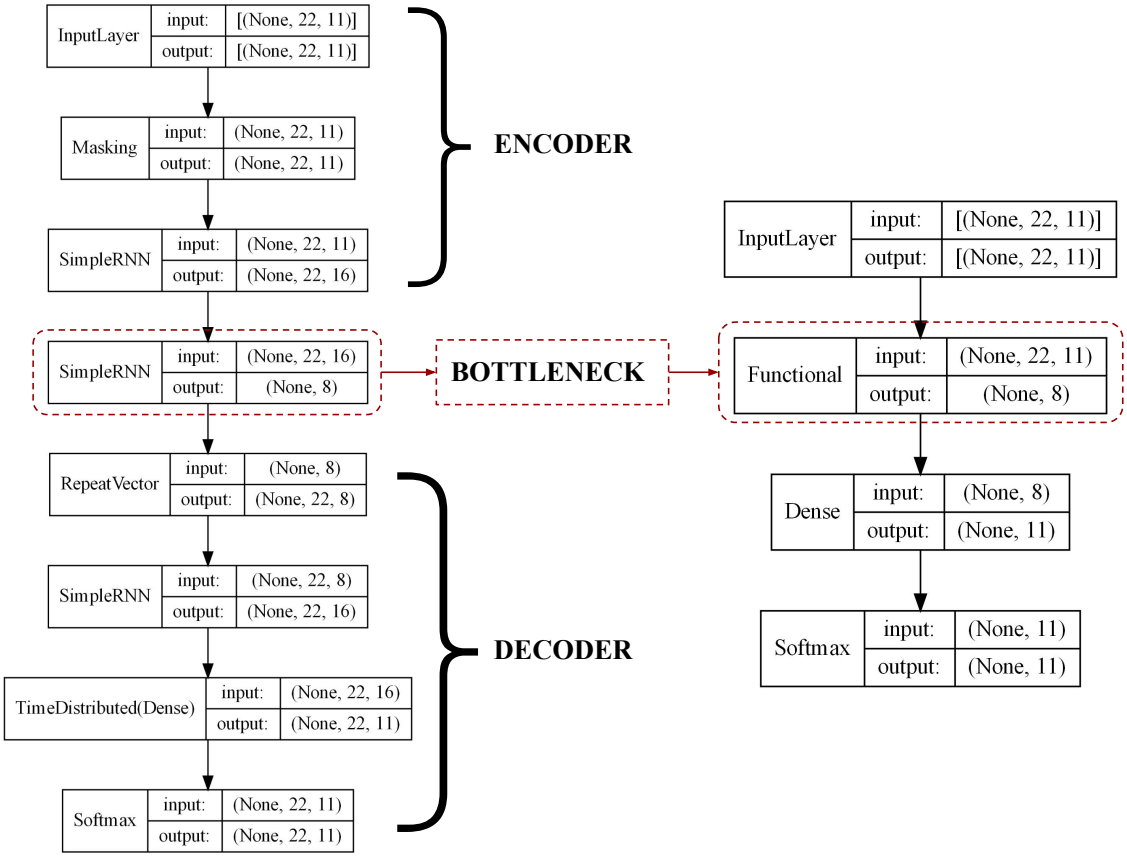

Figure 4.3: autoencoder_arch_1 model architecture. The left panel shows the autoencoder trained in the task of reproducing the inputs. The bottleneck layer learns a 8-dimensional representation of the activity vectors, and it is then used as input of the feedforward neural network shown on the right panel. In the second networks, the `Functional` layer is set to non trainable because node weights have already been defined in the first network.

other hand, it is important to note that the first part only needs to be trained once, and that the number of parameters of the second part is significantly lower than in the other models.

LSTM layers have a significantly higher number of parameters than others, but this only results in a small performances improvement. In our case, since traces are not too long, RNN layers are able to deal with the amount of information available without encountering the short memory problem. For real-world datasets it may be necessary to use more complex layers.

NETWORK NAME	INPUT FEATURES	PREDICTED FEATURES	HIDDEN LAYERS	TRAINABLE PARAMETERS	LOSS
RNN_arch_1	Activity	Activity	`RNN 16`	635	0.841
RNN_arch_2	Activity	Activity	`RNN 32`	1771	0.840
GRU_arch_1	Activity	Activity	`GRU 16`	1579	0.848
GRU_arch_2	Activity	Activity	`GRU 32`	4683	0.836
LSTM_arch_1	Activity	Activity	`LSTM 16`	1979	0.842
LSTM_arch_2	Activity	Activity	`LSTM 32`	5995	0.839
autoencoder_arch_1*	Activity	Activity	`RNN 16` encoder `RNN 8` bottleneck `RNN 16` decoder	99 (1235)	0.836
autoencoder_arch_2*	Activity	Activity	`RNN 32` encoder `RNN 16` bottleneck `RNN 32` decoder	187 (4123)	0.834

* model trained for 200 epochs.

Table 4.1: Architecture exploration summary results. We reported only not overfitted models. Regarding the autoencoder-based architectures, the number of parameters in brackets is the one of the autoencoder network (the one of the left panel in Fig. 4.3), whereas the other is the one of the first part.

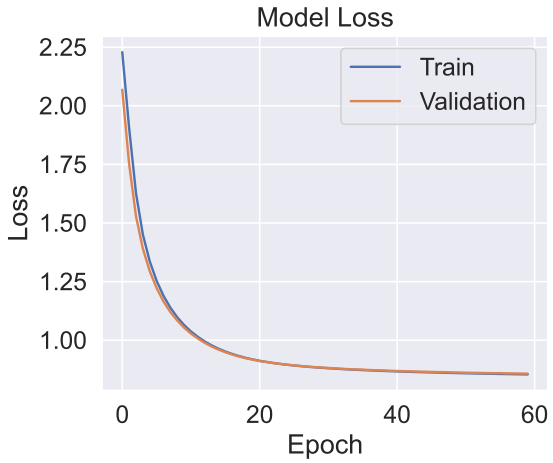

Figure 4.4: RNN_arch_1 loss learning curves on the training and validation set over the epochs.

4.1.2 Features exploration

In this experiment, we investigate the impact of incorporating additional features to neural networks inputs. Since the loss values of the previous experiment were similar, we select RNN_arch_1 as the main fixed architecture for this experiment given the fact that it has the lowest complexity among all.

Non-temporal features do not require to be fed to temporal layers such as `SimpleRNN`, there-

fore we pass them only to the final `Dense` layer of the network. An example of the model architecture with additional non-temporal inputs can be seen in Fig. 4.5. On the contrary, the temporal feature, representing the time difference between activities, needs to be incorporated into temporal layers, just like traces inputs. To achieve this, we concatenate activity vectors and time feature vectors in different ways, depending on the model. An example of a model architecture with time feature inputs can be seen in Fig. 4.6.

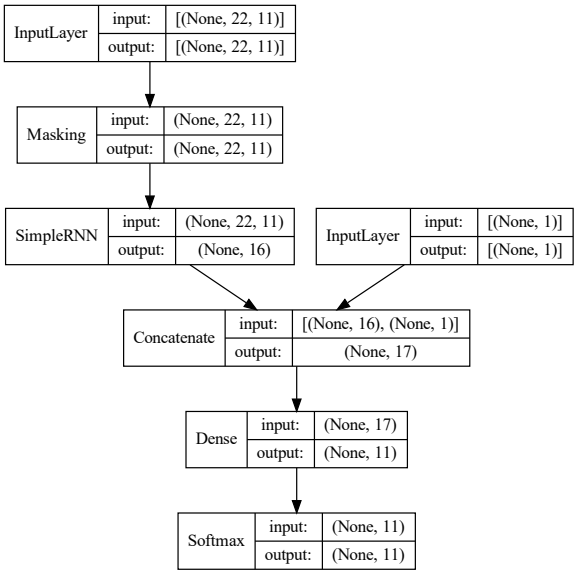

Figure 4.5: feature_pr model architecture. Additional inputs can be inserted using Keras Functional API. The input is concatenated to the output of the `SimpleRNN` layer and the resulting vector is passed to the final layers.

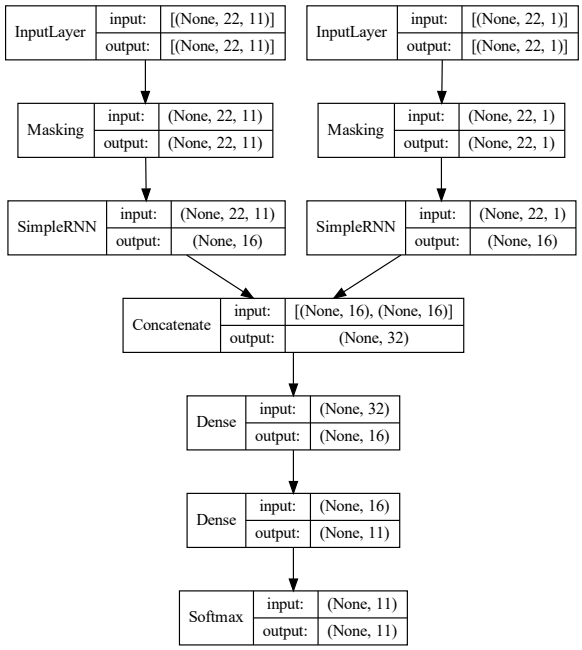

Figure 4.6: feature_time_4 model architecture. We treat the time feature as a temporal feature as well as the activity input. The outputs of the `SimpleRNN` layers are then merged together using a `Concatenate` layer. In this particular network, it is present an additional `Dense` layer before the output layers.

Results

Experiments results indicate that including case attributes in the model training phase generally leads to an improvement in its performance. Specifically, the addition of the PR feature, which has value 0 when the item does not require a PR and value 1 when it is needed, leads to a lower loss function value. This attribute condenses a significant amount of information into a single

binary number. By incorporating it, models become better equipped to identify which activities are likely to occur and which are not. A comparison between confusion matrices of models trained with and without additional input features can be seen in Fig. 4.7. The most important improvement is the classification of the activity "Create Purchase Requisition Item", which is well predicted in the second model, but not in the first. The main difference between process variants concerns materials that do or do not require PR. Most of the additional features carry this information.

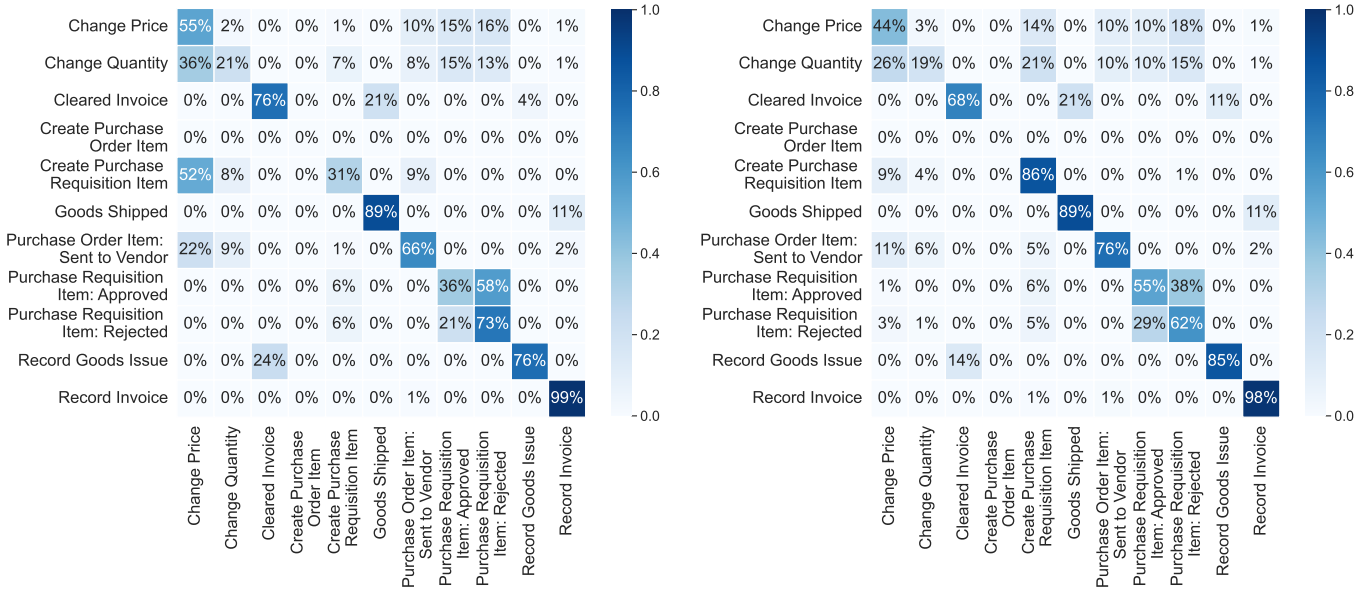

Figure 4.7: Comparison between confusion matrices of models trained with (right panel) and without (left panel) attributes as input. The second confusion matrix shows an important improvement in the prediction of the activity "Create Purchase Requisition Item activity", which is an attribute contained in the majority of the features. Overall the predictions of the second model are more accurate than the ones of the first.

On the other hand, the inclusion of time feature do not result in a noticeable improvement in models performances. This outcome was expected, as in our dataset the temporal attribute is directly correlated with process activities in a standard manner and it does not provide any additional information. In a real-world scenario, for example, an accumulation of delays in the activities execution time can potentially increase the likelihood of failures occurring. Tab. 4.2 shows a summary of the results obtained during experiments involving the addition of non-temporal features, whereas results about the inclusion of time features are provided in Tab. 4.3.

NETWORK NAME	INPUT FEATURES	PREDICTED FEATURES	HIDDEN LAYERS	TRAINABLE PARAMETERS	LOSS
feature_pr	Activity, PR	Activity	`RNN 16`	646	0.792
feature_plant	Activity, Plant	Activity	`RNN 16`	701	0.805
feature_vendor	Activity, Vendor	Activity	`RNN 16`	844	0.787
feature_matgroup	Activity, Mat Group	Activity	`RNN 16`	723	0.791
feature_material	Activity, Mat Number	Activity	`RNN 16`	1735	0.789

Table 4.2: Feature exploration summary results. The number of trainable parameters of the networks is similar and changes only according to the input dimension of the feature used. The model that reaches the lowest loss score is feature_vendor. It is important to notice that attributes share mutual information. For example, the vendor attribute contains information about PR, since some vendors only sell free-pass materials.

NETWORK NAME	INPUT FEATURES	PREDICTED FEATURES	HIDDEN LAYERS	TRAINABLE PARAMETERS	LOSS
feature_time_1	Activity, Timestamp	Activity	`RNN 16` shared	651	0.844
feature_time_2	Activity, Timestamp	Activity	`RNN 16` act `RNN 16` time	811	0.840
feature_time_3	Activity, Timestamp	Activity	`RNN 16` act `RNN 8` time	803	0.847
feature_time_4	Activity, Timestamp	Activity	`RNN 16` act `RNN 16` time `Dense 16` shared	1451	0.824

Table 4.3: Time feature exploration summary results. Adding other temporal layers leads to an increase in the models complexity. In this case, in order to reach a lower score than the vanilla model, it is necessary to add a `Dense` layer as hidden layer. Adding more temporal layers does not result in improving performances, but only in adding complexity.

4.1.3 Multiple features and encoding exploration

Until now, we processed categorical features using one-hot encoding method. One of its main issue is that the resulting vector could be a large sparse matrix. Therefore, we conduct experiments comparing different encoding methods and different input features. Specifically, we focus on four main architecture types: the first one is one-hot encoding-based *fully shared*, the second is one-hot encoding-based *specialized*, the third one is embedding-based *fully shared*, the last one is embedding-based *specialized*. In *fully shared* architectures, we concatenate features together at the beginning and consider them as one vector when applying the encoding methods. In *specialized* architectures, on the other hand, we apply encoding methods to sep-

arate features. We test different configurations of the four main architectures, including variations in the number of nodes and in the dimensionality of the `Embedding` layers. Fig. 4.8 shows an example of the first model architecture type.

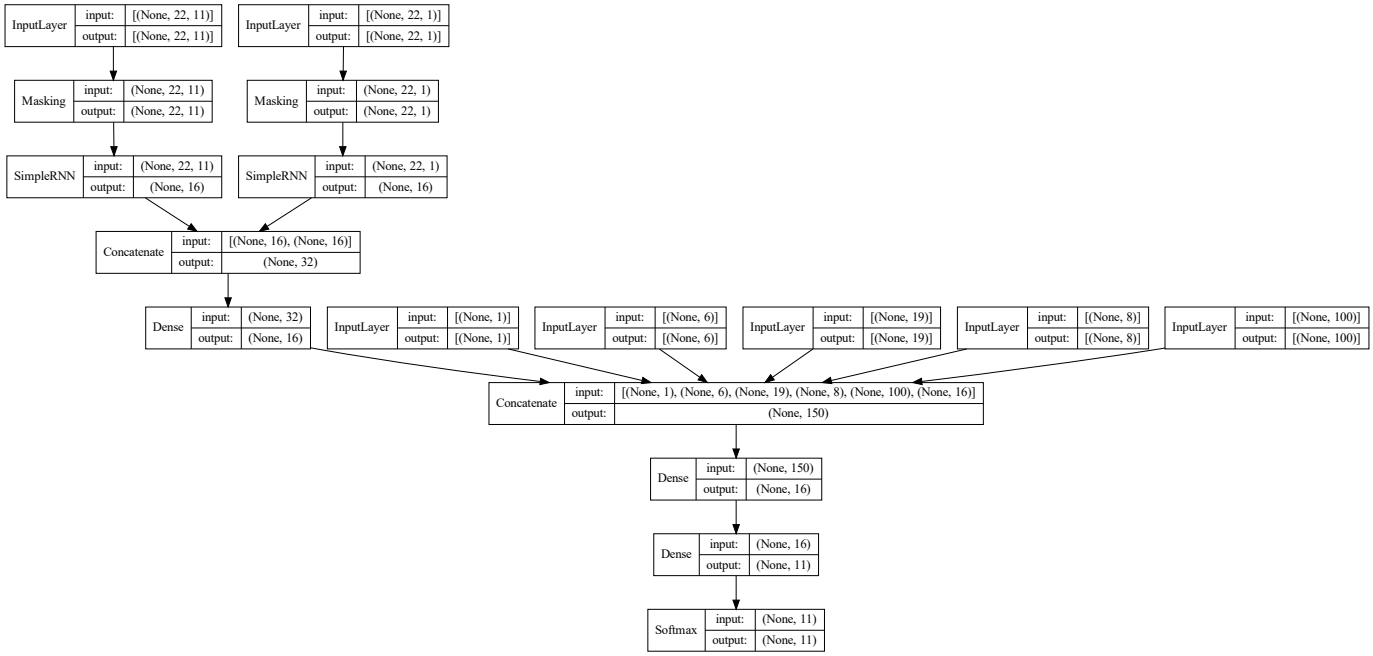

Figure 4.8: One-hot encoding *shared* model architecture. We process temporal features as in the network feature_time_4. The output of the temporal shared `Dense` layer is concatenated along with every case attribute. We pass the resulting vector of size 150 through an intermediate `Dense` layer and then through the usual output layers.

Results

We start by exploring different attributes combinations as input by observing their contribution in the model performances. Initially, we add features that mostly decrease the loss function in the previously experiment. A summary of the results achieved is reported in Tab. 4.4. When adding time and plant features, the loss substantially reduces, whereas it remains stable when adding material group and PR attributes. This may be due to the fact that some attributes share the majority of information, and thus their combination only adds complexity. On the other hand, model multi_feature_5 achieves better results because the plant attribute carries new information and therefore helps the model in the activity prediction.

Regarding the encoding exploration, the main results obtained from the experiments are summarized in Tab. 4.5. Outcomes indicate that one-hot outperforms the use of embedding as encoding method. This finding suggests that in our specific scenario the neural network is capable of effectively handling the one-hot encoded features. This may be due to the fact that the features cardinalities are not excessively large. Therefore models do not need to learn a hidden representation, that can result in increasing complexity. However, it is worth noting that it is possible to design an embedding model that can be trained only once and then reused for prediction tasks. This leads to a lower number of trainable parameters in the subsequent stages of the network, contributing to its efficiency. Moreover, in a real-world scenario the number of unique values for attributes can be thousands. In this case, embedding may be necessary.

NETWORK NAME	INPUT FEATURES	PREDICTED FEATURES	HIDDEN LAYERS	TRAINABLE PARAMETERS	LOSS
multi_feature_1	Activity, Vendor, Mat Number	Activity	`RNN 16`	1944	0.787
multi_feature_2	Activity, Vendor, Mat Number, Time	Activity	`RNN 16` act `RNN 16` time `Dense 16` shared	2760	0.761
multi_feature_3	Activity, Vendor, Mat Number, Time, Mat Group	Activity	`RNN 16` act `RNN 16` time `Dense 16` shared	2848	0.764
multi_feature_4	Activity, Vendor, Mat Number, Time, Mat Group, PR	Activity	`RNN 16` act `RNN 16` time `Dense 16` shared	2859	0.764
multi_feature_5	Activity, Vendor, Mat Number, Time, Mat Group, PR, Plant	Activity	`RNN 16` act `RNN 16` time `Dense 16` shared	2925	0.717

Table 4.4: Multiple feature exploration summary results. The last model, that includes all possible attributes, reaches the lowest loss function value. When adding the time feature, we change the architecture to match the one of feature_time_4. In any case, additional non-temporal attributes are concatenated to the output of the temporal `Dense` layer and the resulting vector is used as input to the last `Dense` layer, as shown in Fig.4.8.

4.2 Multioutput prediction

In a predictive process monitoring task, we are not only interested in the next activity that can occur, but also in the prediction of cases attributes, such as the time at which the next activity will occur, the resource that will perform it, etc. We approach the problem by training our

NETWORK NAME	INPUT FEATURES	PREDICTED FEATURES	HIDDEN LAYERS	TRAINABLE PARAMETERS	LOSS
all_features_emb_1	Activity, PR, Plant, Vendor, Mat Group, Mat Number, Time	Activity	`RNN 16` act `RNN 16` time `Dense 16` shared `Embedding(50)` NT shared	10723	0.721
all_features_emb_2	Activity, PR, Plant, Vendor, Mat Group, Mat Number, Time	Activity	`RNN 16` act `RNN 16` time `Dense 16` shared `Embedding(n_class//2)` NT	8036	0.718
all_features_ohe_1	Activity, PR, Plant, Vendor, Mat Group, Mat Number, Time	Activity	`RNN 16` act `RNN 16` time `Dense 16` shared	3867	0.713
all_features_ohe_2	Activity, PR, Plant, Vendor, Mat Group, Mat Number, Time	Activity	`RNN 16` act `RNN 16` time `Dense 16` shared	5227	0.714

Table 4.5: Feature and encoding exploration summary results. The first model has the highest number of trainable parameters because of the shape of the `Embedding` output, which is (5, 50). Generally, embedding-based networks have a higher number of trainable parameters but this do not results in higher scores.

network in the task of performing multiple output prediction.

We start by predicting the next activity and the time difference of its execution. We tackle the time feature prediction as a regression problem. Therefore this output does not require the final `Softmax` layer but only a `Dense 1` layer. We use the Mean Absolute Error as loss function. Then, we approach the problem of classifying the Maverick Buying in a supervised way. Finally, we introduce the prediction of traces activities encoded as attributes ("Late Shipment", "Late Delivery" and "Late Payment"), along with the three way mismatch anomaly. These are classification predictions and therefore we use an output similar to the activity one, but with a `Sigmoid` activation function as last layer and the *binary cross-entropy* as loss function. We apply an appropriate Temperature Scaling method also to these outputs. An example of a used model architecture is shown in Fig. 4.9.

Results

Tab. 4.6 reports a summary of the loss values obtained. Even if we need to train models for more epochs to reach similar loss values, results obtained by training the network on multiple outputs are lower than those obtained by training the network on a single output. This mainly concerns time prediction. Moreover, single output models tend to overfit more than the others. Learning curves are shown in Fig. 4.10 and Fig. 4.11. We do not report the learning curves for

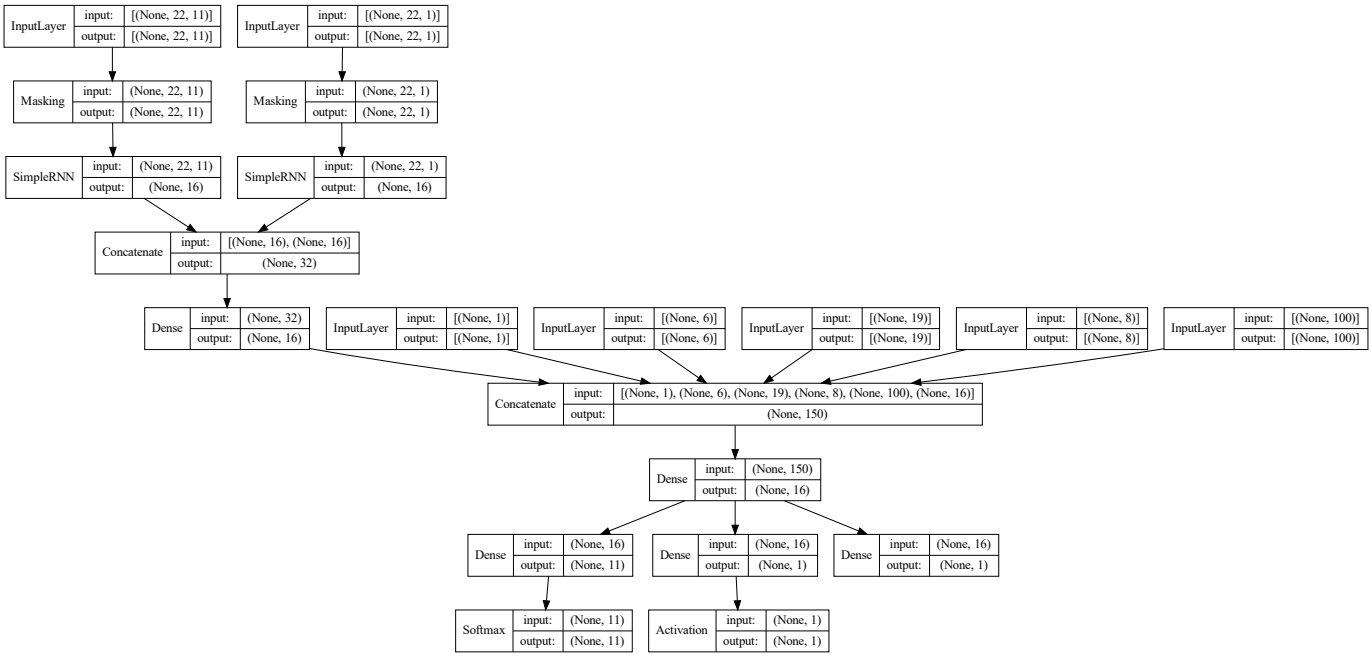

Figure 4.9: Multioutput prediction model architecture. We used all attributes as inputs. The outputs of this model are next activities (`Dense 11` layer), next activity execution time (`Dense 1` layer) and Maverick Buying anomaly (`Dense 1` layer). First and last outputs are passed to `Softmax` and `Sigmoid` activation functions, respectively, because they are classification tasks.

the activity predictions since they are similar to the ones observed for single output models. Fig. 4.12 and Fig. 4.13 display models confusion matrices of the additional prediction tasks.

Learning curves of some target continue to decrease even after 200 epochs. This suggests the need to train models for more epochs, but at risk of overfitting, or to increase the complexity of the network. One of the main problem of this multioutput approach is that adjusting one prediction may result in overfitting another. It can be difficult to find the right model architecture and reach a compromise between all predictions.

Predictions of features related to delay activities and the three way mismatch anomaly are significantly less accurate than others. Regarding the firsts, this may be due to the fact that we did not simulate related dates based on specific vendors or materials, but mostly randomly. The second anomaly, on the other hand, is related to activities "Change Price" and "Change Quantity", which are plant-based, but its ratio is fixed for every material and vendor. In a real-world scenario these characteristics can have a stronger correlation with case attributes.

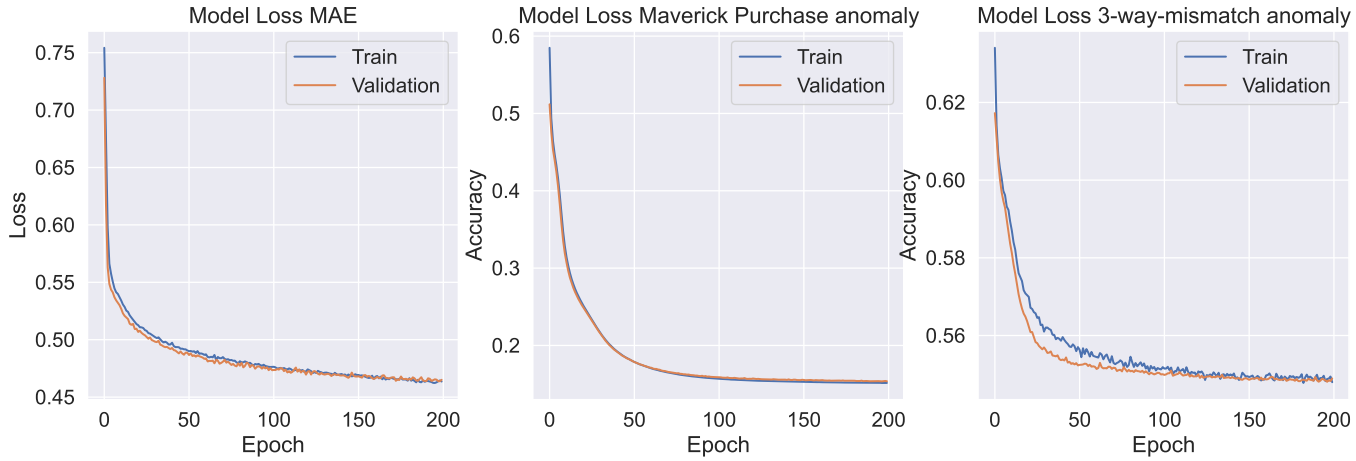

Figure 4.10: multi_output_4 learning curves of time, Maverick Buying and 3-way-mismatch anomaly predictions. The loss function for the first prediction task is MAE, whereas for the lasts we use the binary cross-entropy loss. The loss function of the time prediction continues decreasing after 200 epochs, but increasing the number of parameters of training for more epochs results in overfit.

Figure 4.11: multi_output_4 learning curves of Late Shipment, Late Delivery and Late Payment. The model is trained using binary cross-entropy. For the second prediction task, the model overfit.

NETWORK NAME	PREDICTED FEATURES	TRAINABLE PARAMETERS	ACT LOSS	MAE	MAVP LOSS	TWMS LOSS	LATES LOSS	LATEP LOSS	LATED LOSS
multi_output_1*	Activity, Time	3884	0.720	0.442					
multi_output_2*	Activity, Time, Mav. buying	3901	0.718	0.450	0.159				
multi_output_3*	Activity, Time, Mav. buying, 3-way-mism., Late ship., Late paym., Late del.	4513	0.717	0.454	0.1587	0.533	0.543	0.426	0.466
multi_output_4*	Activity, Time, Mav. buying, 3-way-mism., Late ship., Late paym., Late del.	4771	0.714	0.453	0.156	0.533	0.537	0.425	0.462

* model trained for 200 epochs.

Table 4.6: Multioutput prediction summary results. We test different feature predictions, such as next activity time execution, Maverick Buying, 3-way-mismatch, Late Payment, Late Shipment and Late Delivery, using different model architectures. Models overfit for some prediction tasks.

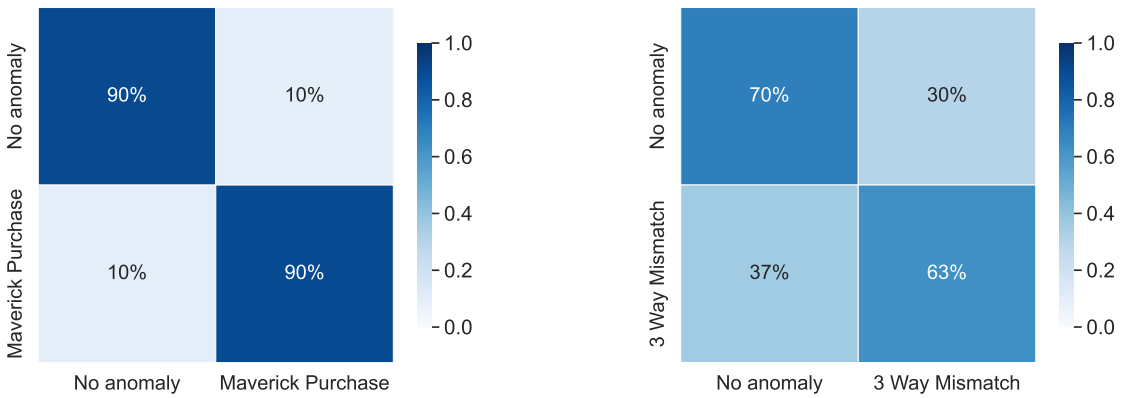

Figure 4.12: multi_output_4 Maverick Buying and Three-way-mismatch confusion matrices.

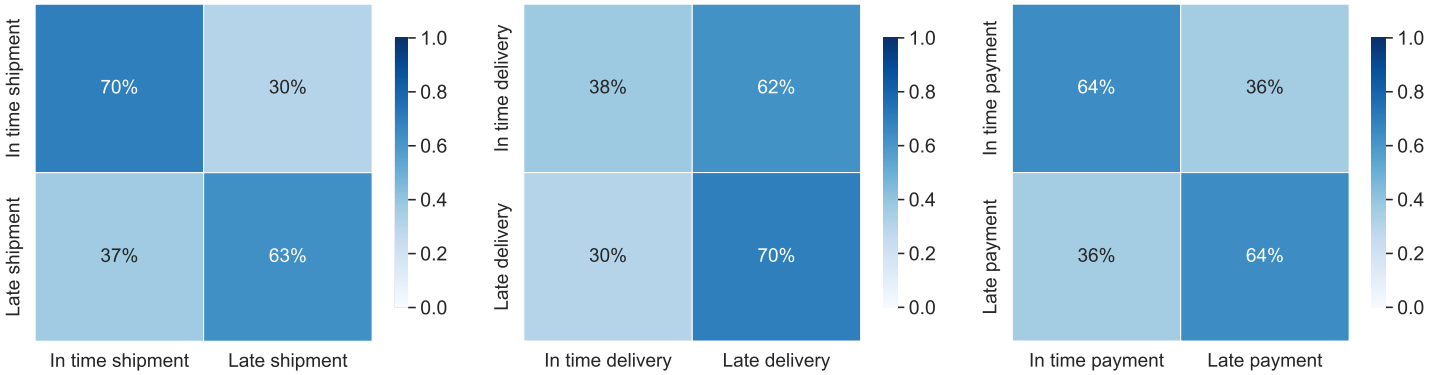

Figure 4.13: multi_output_4 anomalies confusion matrices, in order: Late Shipment, Late Delivery, and Late Payment. Results obtained are worse than in other prediction tasks.

5

Conclusion and further works

The main contribution of this thesis is to illustrate a process mining application, with focus on the utilization of deep learning for predictive process monitoring. The work is part of a larger demo project aimed at showcasing process mining applications and strengths to potential customers.

The dataset presented in this thesis serves as a simplified representation of a real enterprise dataset, providing a foundation for further implementations and explorations. We can add further tables and fields to enhance the complexity of the process and better mimic real-world scenarios. By introducing new fields, it becomes possible to leverage these additional features for deep learning purposes, enabling more comprehensive analysis and prediction tasks. The expansion can also involve incorporating new anomalies, such as random variations and missing data, thereby reflecting the challenges faced in real business environments. Moreover, this dataset can be valuable for training new users on process mining platforms, in particular with regard to extraction and transformation phase. Indeed, this phase, which involves extracting relevant information from raw event logs and transforming it into a suitable format for analysis, is often refined through experience.

In the deep learning experiments conducted, the evaluation approach was centered around the probability distribution rather than solely focusing on the most likely activity prediction. Models tend to predominantly predict the most likely variant, which typically corresponds to the process without anomalies. But this would be useless in a predictive process monitoring operation. By evaluating probability distributions, we focus the attention on the models pre-

dictions rather than on the actual outcome. Moreover, we employed a weighted loss function which assigns higher weights to the rarer activities during training. By giving more emphasis to these activities, models are encouraged to pay closer attention to them and provide more accurate predictions for their occurrence. The outcome of using a weighted loss function is that predictions exhibit a greater variance in activity prediction compared to when a standard loss function is used. By emphasizing the probability distribution and incorporating a weighted loss function, models become more sensitive to the presence of anomalies and can provide a more comprehensive understanding of the underlying process dynamics. This approach enables a more nuanced analysis and prediction, allowing for better anomaly detection and mitigation in real-world business processes.

Models developed for the activity prediction task demonstrate the capability to capture process anomalies in a supervised manner, particularly anomalies that manifest as deviations in the order of activities. This outcome is significant as such anomalies are commonly encountered in real-world business processes and are often detected through manual inspection by human experts. Results achieved in the multioutput prediction task are not optimal, especially those concerning the prediction of delay activities and three way mismatch. This is because of the data used, which were not specifically designed for this type of prediction. Further studies need to be carried out in the field of multioutput prediction, especially regarding model architectures and data encoding. However, the intention was to show the possibility of creating models capable of predicting more than one feature at the same time, even of a different nature (numerical and categorical). We have shown that it is possible to exploit process knowledge and encode information contained in activities into classes. In general, each process must be analysed in order to create the most suitable predictive monitoring model.

The ability of models to detect and predict anomalies with a high degree of confidence provides valuable insights and contributes to process improvement efforts. It enables the creation of a simple and straightforward indicator that can be integrated into a process mining platform, serving as an alert mechanism for end users. The indicator serves as a valuable tool for decision-making, allowing users to take appropriate actions to address potential anomalies before they escalate or impact the overall process performance. By leveraging the model's predictions, users can be promptly notified about the likelihood of process anomalies occurring in ongoing or open traces. An example of an application of a predictive process monitoring model is shown in Fig.5.1 and Fig.5.2. These figures represent a table contained in the Celonis platform. We perform multioutput predictions on open traces using a model presented in Sec.4.2. Then, we load the results in the process mining platform and display them using tables that the platform

provides. An end user can use these tables in order to visualize the risk levels of open traces. If an order has a high anomalies ratio associated anomalies, the user can select it and decide to send an email or message to the supervisor. Moreover, it is possible to configure action flows that are automatically triggered whenever an order exceeds a certain risk threshold. Using process mining platforms, the possibilities for anomaly management are almost endless.

Werks	Ebeln	Ebelp	Next Activity Time	Purchase Requisition Item Rejected	Change Price	Change Quantity
IN10	1005439	2	2 days	63.10 %	5.30 %	7.50 %
CN10	1005442	3	1 days	59.00 %	2.30 %	3.50 %
IN10	1005437	2	3 days	56.90 %	1.40 %	1.60 %
IN10	1005456	1	1 days	54.20 %	7.80 %	6.30 %
IN10	1005438	2	2 days	52.00 %	1.40 %	1.50 %
CN10	1005463	2	1 days	50.30 %	5.80 %	7.60 %
US10	1005451	1	1 days	49.30 %	1.10 %	1.70 %

Figure 5.1: Celonis table containing next activities prediction. We decide to report only anomalous activities and the next time in day at which the activity may be performed.

Send Email (2)

Werks	Ebeln	Ebelp	Maverick Purcha...	Threewaymisma...	Late Delivery	Late Payment	Late Shipment
IN10	1005301	3	100.00 %	52.30 %	21.30 %	16.10 %	25.40 %
CN10	1005295	2	100.00 %	49.40 %	18.00 %	15.30 %	32.70 %
IN10	1005358	3	100.00 %	46.60 %	17.40 %	11.60 %	17.70 %
CN10	1005329	2	100.00 %	46.30 %	17.10 %	0.10 %	15.00 %
US10	1005303	2	100.00 %	45.90 %	16.40 %	0.10 %	19.90 %
IN10	1005373	1	100.00 %	45.80 %	17.90 %	18.90 %	9.20 %
IN10	1005373	2	100.00 %	44.50 %	20.70 %	25.10 %	8.70 %

Figure 5.2: Celonis table containing anomalies prediction. It is shown the possibility to apply action and trigger action flows that allows companies to solve anomalies instantly.

One of the most important study that can be conducted is approaching the problem through unsupervised techniques. In this way, it becomes possible to partially eliminate the reliance on complex and incomplete process documentation, pre-defined labels or prior knowledge of expected process behavior. This approach is particularly valuable in situations where the process documentation is scarce or not up to date, as it can adapt and learn from the data itself. Unsupervised techniques offer the advantage of flexibility and adaptability, allowing the system to continuously learn and improve as new data stream in. Mastery of these techniques enables the identification of anomalies in real-time or near real-time, providing early detection and proactive measures to address potential issues before they escalate. Implementing unsupervised techniques in process mining opens up new possibilities for automating anomaly detection, reducing reliance on manual analysis and improving the overall efficiency and effectiveness of

process monitoring. It empowers organizations to gain valuable insights from their data, enhance decision-making processes, and ensure smooth and optimized operations.

Further work can be conducted to study the social network aspect of the process. By incorporating additional information about employees, organizational structure, roles, and other relevant factors, it becomes possible to analyze the interactions and relationships among individuals in the company. One valuable approach is to examine the activity executives and use that information to construct a social network of the company. In large companies with multiple branches and operations in different countries, understanding the social network can be particularly crucial. It allows for the identification of communication gaps, inefficiencies, or structural issues that may impede effective collaboration and hinder the smooth execution of processes. By uncovering these bottlenecks and failures, organizations can take proactive measures to address them and optimize their operations.

Process mining is a multidisciplinary field that requires the involvement of various experts with different backgrounds and knowledge. In a business context, it is crucial to incorporate all available information into the process investigation in order to fully leverage the technologies potential. This information can be derived from the process model itself, as well as from the company dataset or through interviews with process owners. This comprehensive approach is necessary because modern processes and supply chains are highly complex, involving numerous entities. Furthermore, many companies have not yet fully embraced recent technological advancements and process standardization practices. The increasing complexity of modern processes necessitates the modernization of existing processes and systems to handle and analyze the vast amount of information present in today's databases. This thesis addresses these challenges by highlighting the importance of integrating process mining techniques with all the available knowledge and points out the need for companies to adopt advanced technologies to support efficient and effective analysis of business processes.

References

[1] F. Almeida, J. Duarte Santos, and J. Augusto Monteiro, "The challenges and opportunities in the digitalization of companies in a post-covid-19 world," *IEEE Engineering Management Review*, vol. 48, no. 3, pp. 97–103, 2020.

[2] B. Obrenovic, J. Du, D. Godinic, D. Tsoy, M. A. S. Khan, and I. Jakhongirov, "Sustaining enterprise operations and productivity during the covid-19 pandemic: "enterprise effectiveness and sustainability model"," *Sustainability*, vol. 12, no. 15, 2020. [Online]. Available: https://www.mdpi.com/2071-1050/12/15/5981

[3] C. Guérin, "What's next for the raw material crisis and how companies can avoid its downfalls during times of uncertainty," https://www.nasdaq.com/articles/whats-next-for-the-raw-material-crisis-and-how-companies-can-avoid-its-downfalls-during, accessed: 2023-05-24.

[4] T. P. Wiśniewski, "Investigating divergent energy policy fundamentals: Warfare assessment of past dependence on russian energy raw materials in europe," *Energies*, vol. 16, no. 4, 2023. [Online]. Available: https://www.mdpi.com/1996-1073/16/4/2019

[5] P. C. Godfrey, "The relationship between corporate philanthropy and shareholder wealth: A risk management perspective," *Academy of management review*, vol. 30, no. 4, pp. 777–798, 2005.

[6] T. Hahn, F. Figge, J. Pinkse, and L. Preuss, "Trade-offs in corporate sustainability: You can't have your cake and eat it," pp. 217–229, 2010.

[7] T. Hahn, F. Figge, J. A. Aragón-Correa, and S. Sharma, "Advancing research on corporate sustainability: Off to pastures new or back to the roots?" *Business & Society*, vol. 56, no. 2, pp. 155–185, 2017.

[8] C. Schreiber, "Automated sustainability compliance checking using process mining and formal logic," in *Proceedings of the 7th international conference on ICT for sustainability*, 2020, pp. 181–184.

[9] L. Reinkemeyer, "Status and future of process mining: from process discovery to process execution," in *Process Mining Handbook*. Springer, 2022, pp. 405–415.

[10] W. M. van der Aalst, "Process mining: a 360 degree overview," in *Process Mining Handbook*. Springer, 2022, pp. 3–34.

[11] N. Candito, "How inefficient processes are hurting your company," https://www.entrepreneur.com/growing-a-business/how-inefficient-processes-are-hurting-your-company/286084, accessed: 2023-05-24.

[12] T. Takebayashi, H. Tsuda, T. Hasebe, and R. Masuoka, "Data loss prevention technologies," *Fujitsu Scientific and Technical Journal*, vol. 46, no. 1, pp. 47–55, 2010.

[13] D. E. A. Sanders, "The modelling of extreme events," *British Actuarial Journal*, vol. 11, no. 3, p. 519–557, 2005.

[14] R. Bean, "Why culture is the greatest barrier to data success," https://sloanreview.mit.edu/article/why-culture-is-the-greatest-barrier-to-data-success/, accessed: 2023-05-24.

[15] M. Siek and R. Mukti, "Business process mining from e-commerce event web logs: Conformance checking and bottleneck identification," in *IOP Conference Series: Earth and Environmental Science*, vol. 729, no. 1. IOP Publishing, 2021, p. 012133.

[16] S. Park and Y. S. Kang, "A study of process mining-based business process innovation," *Procedia Computer Science*, vol. 91, pp. 734–743, 2016.

[17] W. Van Der Aalst, "Process mining: Overview and opportunities," *ACM Transactions on Management Information Systems (TMIS)*, vol. 3, no. 2, pp. 1–17, 2012.

[18] W. M. Van Der Aalst and S. Dustdar, "Process mining put into context," *IEEE Internet Computing*, vol. 16, no. 1, pp. 82–86, 2012.

[19] T. B. H. Tu and M. Song, "Analysis and prediction cost of manufacturing process based on process mining," in *2016 International Conference on Industrial Engineering, Management Science and Application (ICIMSA)*. IEEE, 2016, pp. 1–5.

[20] W. M. Van der Aalst, M. H. Schonenberg, and M. Song, "Time prediction based on process mining," *Information systems*, vol. 36, no. 2, pp. 450–475, 2011.

[21] N. Mehdiyev and P. Fettke, "Explainable artificial intelligence for process mining: A general overview and application of a novel local explanation approach for predictive process monitoring," *Interpretable Artificial Intelligence: A Perspective of Granular Computing*, pp. 1–28, 2021.

[22] W. M. van der Aalst, "Process mining and simulation: A match made in heaven!" in *SummerSim*, 2018, pp. 4–1.

[23] O. A. Johnson, T. Ba Dhafari, A. Kurniati, F. Fox, and E. Rojas, "The clearpath method for care pathway process mining and simulation," in *Business Process Management Workshops: BPM 2018 International Workshops, Sydney, NSW, Australia, September 9-14, 2018, Revised Papers 16*. Springer, 2019, pp. 239–250.

[24] W. Abohamad, A. Ramy, and A. Arisha, "A hybrid process-mining approach for simulation modeling," in *2017 Winter Simulation Conference (WSC)*. IEEE, 2017, pp. 1527–1538.

[25] W. M. P. van der Aalst, *Process Mining: Data Science in Action*, 2nd ed. Heidelberg: Springer, 2016.

[26] M. Heravizadeh, J. Mendling, and M. Rosemann, "Root cause analysis in business processes," 2008.

[27] W. Aalst, K. Hee, J. M. Van der Werf, and M. Verdonk, "Auditing 2.0: Using process mining to support tomorrow's auditor," *Computer*, vol. 43, pp. 90 – 93, 04 2010.

[28] J. Evermann, J. Rehse, and P. Fettke, "Predicting process behaviour using deep learning," *CoRR*, vol. abs/1612.04600, 2016. [Online]. Available: http://arxiv.org/abs/1612.04600

[29] D. M. V. Sato, S. C. De Freitas, J. P. Barddal, and E. E. Scalabrin, "A survey on concept drift in process mining," *ACM Comput. Surv.*, vol. 54, no. 9, oct 2021. [Online]. Available: https://doi.org/10.1145/3472752

[30] H. M. Marin-Castro and E. Tello-Leal, "Event log preprocessing for process mining: a review," *Applied Sciences*, vol. 11, no. 22, p. 10556, 2021.

[31] M. F. Sani, "Preprocessing event data in process mining," in *CAiSE (Doctoral Consortium)*, 2020, pp. 1–10.

[32] W. Van Der Aalst, A. Adriansyah, A. K. A. De Medeiros, F. Arcieri, T. Baier, T. Blickle, J. C. Bose, P. Van Den Brand, R. Brandtjen, J. Buijs *et al.*, "Process mining manifesto," in *Business Process Management Workshops: BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I 9*. Springer, 2012, pp. 169–194.

[33] W. Aalst, "Process mining: Overview and opportunities," *ACM Transactions on Management Information Systems*, vol. 3, pp. 7.1–7.17, 07 2012.

[34] D. B. Mirsu, "Monitoring help desk process using kpi," in *Soft Computing Applications: Proceedings of the 5th International Workshop Soft Computing Applications (SOFA)*. Springer, 2013, pp. 637–647.

[35] W. Pan and H. Wei, "Research on key performance indicator (kpi) of business process," in *2012 Second International Conference on Business Computing and Global Informatization*, 2012, pp. 151–154.

[36] Y.-C. Tsai and Y.-T. Cheng, "Analyzing key performance indicators (kpis) for e-commerce and internet marketing of elderly products: A review," *Archives of gerontology and geriatrics*, vol. 55, no. 1, pp. 126–132, 2012.

[37] F. Veit, J. Geyer-Klingeberg, J. Madrzak, M. Haug, and J. Thomson, "The proactive insights engine: Process mining meets machine learning and artificial intelligence." in *BPM (Demos)*, 2017.

[38] G. H. John, *Enhancements to the data mining process*. stanford university, 1997.

[39] M. S. Garver, "Using data mining for customer satisfaction research," *Marketing Research*, vol. 14, no. 1, p. 8, 2002.

[40] A. Alibasic and T. Popovic, "Applying natural language processing to analyze customer satisfaction," in *2021 25th International Conference on Information Technology (IT)*. IEEE, 2021, pp. 1–4.

[41] R. Caruana and A. Niculescu-Mizil, "Data mining in metric space: an empirical analysis of supervised learning performance criteria," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004, pp. 69–78.

[42] V. Nasteski, "An overview of the supervised machine learning methods," *Horizons. b*, vol. 4, pp. 51–62, 2017.

[43] N. Grira, M. Crucianu, and N. Boujemaa, "Unsupervised and semi-supervised clustering: a brief survey," *A review of machine learning techniques for processing multimedia content*, vol. 1, no. 2004, pp. 9–16, 2004.

[44] S. Agrawal and J. Agrawal, "Survey on anomaly detection using data mining techniques," *Procedia Computer Science*, vol. 60, pp. 708–713, 2015.

[45] A. C. Kumar, "Analysis of unsupervised dimensionality reduction techniques," *Computer science and information systems*, vol. 6, no. 2, pp. 217–227, 2009.

[46] K. Fukunaga and R. R. Hayes, "Estimation of classifier performance," *IEEE transactions on pattern analysis and machine intelligence*, vol. 11, no. 10, pp. 1087–1101, 1989.

[47] J. Lever, "Classification evaluation: It is important to understand both what a classification metric expresses and what it hides," *Nature methods*, vol. 13, no. 8, pp. 603–605, 2016.

[48] N. Thai-Nghe, Z. Gantner, and L. Schmidt-Thieme, "Cost-sensitive learning methods for imbalanced data," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, 2010, pp. 1–8.

[49] M. Hossin and M. N. Sulaiman, "A review on evaluation metrics for data classification evaluations," *International journal of data mining & knowledge management process*, vol. 5, no. 2, p. 1, 2015.

[50] A. Luque, A. Carrasco, A. Martín, and A. de Las Heras, "The impact of class imbalance in classification performance metrics based on the binary confusion matrix," *Pattern Recognition*, vol. 91, pp. 216–231, 2019.

[51] R. Benedetti, "Scoring rules for forecast verification," *Monthly Weather Review*, vol. 138, no. 1, pp. 203–211, 2010.

[52] G. W. Brier *et al.*, "Verification of forecasts expressed in terms of probability," *Monthly weather review*, vol. 78, no. 1, pp. 1–3, 1950.

[53] V. Pasquadibisceglie, A. Appice, G. Castellano, and D. Malerba, "Using convolutional neural networks for predictive process analytics," in *2019 International Conference on Process Mining (ICPM)*, 2019, pp. 129–136.

[54] E. Obodoekwe, X. Fang, and K. Lu, "Convolutional neural networks in process mining and data analytics for prediction accuracy," *Electronics*, vol. 11, p. 2128, 07 2022.

[55] N. Navarin, B. Vincenzi, M. Polato, and A. Sperduti, "Lstm networks for data-aware remaining time prediction of business process instances," 2017.

[56] R. Galanti, B. Coma-Puig, M. de Leoni, J. Carmona, and N. Navarin, "Explainable predictive process monitoring," in *2020 2nd International Conference on Process Mining (ICPM)*. IEEE, 2020, pp. 1–8.

[57] N. Mehdiyev and P. Fettke, "Explainable artificial intelligence for process mining: A general overview and application of a novel local explanation approach for predictive process monitoring," *Interpretable Artificial Intelligence: A Perspective of Granular Computing*, pp. 1–28, 2021.

[58] E. Rama-Maneiro, J. C. Vidal, and M. Lama, "Deep learning for predictive business process monitoring: Review and benchmark," *CoRR*, vol. abs/2009.13251, 2020. [Online]. Available: https://arxiv.org/abs/2009.13251

[59] M. Hinkka, T. Lehto, and K. Heljanko, "Exploiting event log data-attributes in RNN based prediction," *CoRR*, vol. abs/1904.06895, 2019. [Online]. Available: http://arxiv.org/abs/1904.06895

[60] N. Tax, I. Verenich, M. L. Rosa, and M. Dumas, "Predictive business process monitoring with LSTM neural networks," in *Advanced Information Systems Engineering*. Springer International Publishing, 2017, pp. 477–492. [Online]. Available: https://doi.org/10.1007%2F978-3-319-59536-8_30

[61] M. Camargo, M. Dumas, and O. González-Rojas, "Learning accurate lstm models of business processes," in *Business Process Management: 17th International Conference, BPM 2019, Vienna, Austria, September 1–6, 2019, Proceedings 17*. Springer, 2019, pp. 286–302.

[62] D. A. Neu, J. Lahann, and P. Fettke, "A systematic literature review on state-of-the-art deep learning methods for process prediction," *CoRR*, vol. abs/2101.09320, 2021. [Online]. Available: https://arxiv.org/abs/2101.09320

[63] N. D. Mauro, A. Appice, and T. M. A. Basile, "Activity prediction of business process instances with inception CNN models," in *Lecture Notes in Computer Science*. Springer International Publishing, 2019, pp. 348–361. [Online]. Available: https://doi.org/10.1007%2F978-3-030-35166-3_25

[64] A. Metzger and A. Neubauer, "Considering non-sequential control flows for process prediction with recurrent neural networks," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2018, pp. 268–272.

[65] J. Lahann, P. Pfeiffer, and P. Fettke, "Lstm-based anomaly detection of process instances: Benchmark and tweaks," in *International Conference on Process Mining Workshops*, 2022.

[66] T. Nolle, A. Seeliger, and M. Mühlhäuser, "Unsupervised anomaly detection in noisy business process event logs using denoising autoencoders," in *Discovery Science: 19th International Conference, DS 2016, Bari, Italy, October 19–21, 2016, Proceedings 19*. Springer, 2016, pp. 442–456.

[67] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow.* " O'Reilly Media, Inc.", 2022.

[68] I. Saito, K. Nishida, K. Nishida, and J. Tomita, "Abstractive summarization with combination of pre-trained sequence-to-sequence and saliency models," *arXiv preprint arXiv:2003.13028*, 2020.

[69] M. Boussakssou, H. Ezzikouri, and M. Erritali, "Chatbot in arabic language using seq to seq model," *Multimedia Tools and Applications*, vol. 81, no. 2, pp. 2859–2871, 2022.

[70] K. Sun, T. Qian, X. Chen, and M. Zhong, "Context-aware seq2seq translation model for sequential recommendation," *Information Sciences*, vol. 581, pp. 60–72, 2021.

[71] L. Dong, S. Xu, and B. Xu, "Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition," in *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2018, pp. 5884–5888.

[72] Y. Su, R. Lin, and C.-C. J. Kuo, "Tree-structured multi-stage principal component analysis (tmpca): theory and applications," *Expert Systems with Applications*, vol. 118, pp. 355–364, 2019.

[73] H. J. Kim, S. E. Hong, and K. J. Cha, "seq2vec: Analyzing sequential data using multi-rank embedding vectors," *Electronic Commerce Research and Applications*, vol. 43, p. 101003, 2020.

[74] Y. Sua, R. Lina, and C.-C. J. Kuoa, "On tree-structured multi-stage principal component analysis (tmpca) for text classification," *arXiv preprint arXiv:1807.08228*, 2018.

[75] L. R. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, pp. 64–67, 2001.

[76] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[77] S. Hochreiter, "Recurrent neural net learning and vanishing gradient," *International Journal Of Uncertainity, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 2, pp. 107–116, 1998.

[78] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[79] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[80] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," *Neural computation*, vol. 31, no. 7, pp. 1235–1270, 2019.

[81] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing lstm language models," *arXiv preprint arXiv:1708.02182*, 2017.

[82] A. Graves, N. Jaitly, and A.-r. Mohamed, "Hybrid speech recognition with deep bidirectional lstm," in *2013 IEEE workshop on automatic speech recognition and understanding*. IEEE, 2013, pp. 273–278.

[83] G. Tiwari, A. Sharma, A. Sahotra, and R. Kapoor, "English-hindi neural machine translation-lstm seq2seq and convs2s," in *2020 International Conference on Communication and Signal Processing (ICCSP)*. IEEE, 2020, pp. 871–875.

[84] J. Wang, L.-C. Yu, K. R. Lai, and X. Zhang, "Dimensional sentiment analysis using a regional cnn-lstm model," in *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)*, 2016, pp. 225–230.

[85] R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (gru) neural networks," in *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*. IEEE, 2017, pp. 1597–1600.

[86] D. Wang, J. Su, and H. Yu, "Feature extraction and analysis of natural language processing for deep learning english language," *IEEE Access*, vol. 8, pp. 46 335–46 345, 2020.

[87] S. Khandelwal, B. Lecouteux, and L. Besacier, "Comparing gru and lstm for automatic speech recognition," Ph.D. dissertation, LIG, 2016.

[88] Y. Hu, A. Huber, J. Anumula, and S.-C. Liu, "Overcoming the vanishing gradient problem in plain recurrent networks," *arXiv preprint arXiv:1801.06105*, 2018.

[89] Y. Wang, H. Yao, and S. Zhao, "Auto-encoder based dimensionality reduction," *Neurocomputing*, vol. 184, pp. 232–242, 2016.

[90] W. Wang, Y. Huang, Y. Wang, and L. Wang, "Generalized autoencoder: A neural network framework for dimensionality reduction," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2014, pp. 490–497.

[91] L. Gondara, "Medical image denoising using convolutional denoising autoencoders," in *2016 IEEE 16th international conference on data mining workshops (ICDMW)*. IEEE, 2016, pp. 241–246.

[92] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." *Journal of machine learning research*, vol. 11, no. 12, 2010.

[93] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1096–1103.

[94] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special lecture on IE*, vol. 2, no. 1, pp. 1–18, 2015.

[95] Z. Chen, C. K. Yeo, B. S. Lee, and C. T. Lau, "Autoencoder-based network anomaly detection," in *2018 Wireless telecommunications symposium (WTS)*. IEEE, 2018, pp. 1–5.

[96] D. Gong, L. Liu, V. Le, B. Saha, M. R. Mansour, S. Venkatesh, and A. v. d. Hengel, "Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1705–1714.

[97] L. Lin, L. Wen, and J. Wang, "Mm-pred: A deep predictive model for multi-attribute event sequence," in *Proceedings of the 2019 SIAM international conference on data mining*. SIAM, 2019, pp. 118–126.

[98] A. Khan, H. Le, K. Do, T. Tran, A. Ghose, H. Dam, and R. Sindhgatta, "Memory-augmented neural networks for predictive process analytics," *arXiv preprint arXiv:1802.00938*, 2018.

[99] L. Trautmann and R. Lasch, "Smart contracts in the context of procure-to-pay," *Smart and Sustainable Supply Chain and Logistics–Trends, Challenges, Methods and Best Practices: Volume 1*, pp. 3–23, 2020.

[100] D. Essex, "procure to pay (p2p)," https://www.techtarget.com/searcherp/definition/procure-to-pay-P2P, accessed: 2023-05-27.

[101] G. Schuh, T. Potente, C. Thomas, and F. Brambring, "Approach for reducing data inconsistencies in production control," in *Enabling Manufacturing Competitiveness and Economic Sustainability: Proceedings of the 5th International Conference on Changeable, Agile, Reconfigurable and Virtual Production (CARV 2013), Munich, Germany, October 6th-9th, 2013*. Springer, 2014, pp. 347–351.

[102] M. Jans, M. Alles, and M. Vasarhelyi, "The case for process mining in auditing: Sources of value added and areas of application," *International Journal of Accounting Information Systems*, vol. 14, no. 1, pp. 1–20, 2013.

[103] Overview about tables in sap. [Online]. Available: https://www.testingbrain.com/sap/data-dictionary-tutorial/what-is-tables-in-sap-types-creation.html

[104] K. Karjalainen and E. Van Raaij, "An empirical test of contributing factors to different forms of maverick buying," *Journal of Purchasing and Supply Management*, vol. 17, no. 3, pp. 185–197, 2011.

[105] A. B. Soliman, K. Eissa, and S. R. El-Beltagy, "Aravec: A set of arabic word embedding models for use in arabic nlp," *Procedia Computer Science*, vol. 117, pp. 256–265, 2017.

[106] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras

[107] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *International conference on machine learning*. PMLR, 2017, pp. 1321–1330.